前端开发笔记本

Li Xinyang

目錄

前端开发笔记本

gitter join chat

点击这里开始在 GitBook 阅读！

点击这里开始在极客学院 Wiki 阅读！（中国大陆访问速度较快）

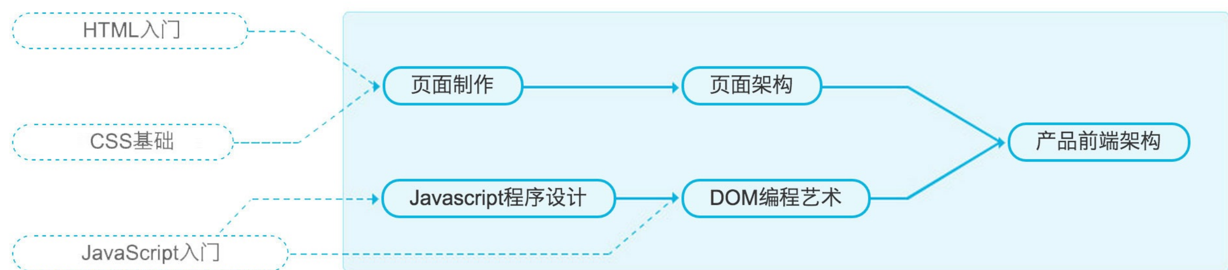
前端开发笔记本的 GitHub 项目地址在这里。如果你觉得这个项目不错，请点击 **Star** 一下，您的支持是我最大的动力。

Star

Watch

Fork

Download



《前端笔记本》涵盖了 Web 前端开发所需的全部基本知识以及所对应的学习路径。它并不能当做一本完整的学习材料来进行前端的学习，因为在有限的篇幅中无法深入的展开每一个单独的知识点。它更适合作为一个学习清单或者是查询手册，结合其他在各个方面更专业的图书或者官方文档来进行同步学习。在学习过程中为了能达到最佳的学习效果，也可将每个技术点实现并进行适当的拓展。

写作进程

第一版草稿 完成时间 1507252244

章节	名称	进程
第一章	页面制作	100%
第二章	JavaScript 程序设计	100%
第三章	DOM 编程	100%
第四章	页面构架	100%
第五章	前端产品构架	100%

成就

1508311803 - 100 Stars: @dszls, thank you!

1511071358 - 200 Stars: @cschenchen

1511222144 - 300 Stars: @lujun9972

1512102333 - 400 Stars: @Niefee

1601231619 - 500 Stars: @wangdsh

1603261510 - 600 Stars: @lty2226262

1605170450 - 700 Stars: @eqzcy

1608010000 - 800 Stars: @BeOker

1610050000 - 900 Stars: @but0n

1612210000 - 1k Stars: @arcobalenoi27 :tada: :tada: :tada:

相关链接

- NEC {N: nice, E: easy, C: css;}

声明

前端笔记本大纲及内容由网易云课堂前端专业改编

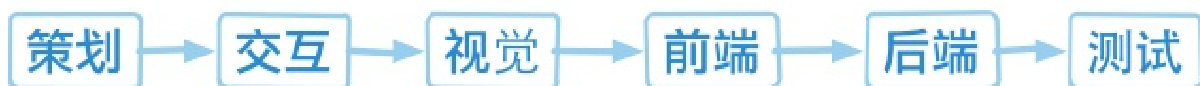
前端工程师概述

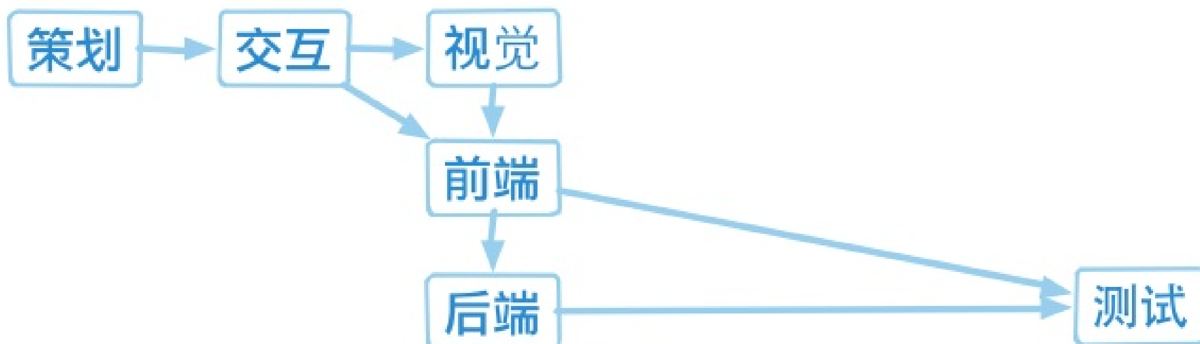
[TOC]

网页发展史

Web 1.0 -> Web 2.0（基于 Ajax） -> Web 3.0 （基于 HTML5）

网站开发协作流程





前端职责

视觉稿（配色图标距离空间） + 交互稿（用户逻辑） = UI（用户界面）

视觉稿 -> 页面制作

交互稿 -> 页面逻辑开发

所需技能

- DOM (操作 HTML 及 CSS 的接口)
- JavaScript (定义页面互动)
- CSS (定义页面样式)
- HTML (定义页面结果)

- Photoshop (取图)

Table of Contents *generated with [DocToc](https://github.com/thlorenz/doctoc)*

Photoshop

切图 从设计稿中切除网页的素材并在代码中引入图片 (复杂的图片或者解决兼容问题)

```
// 设计稿 (*.psd) -> 产出物 (*png, *.jpg)

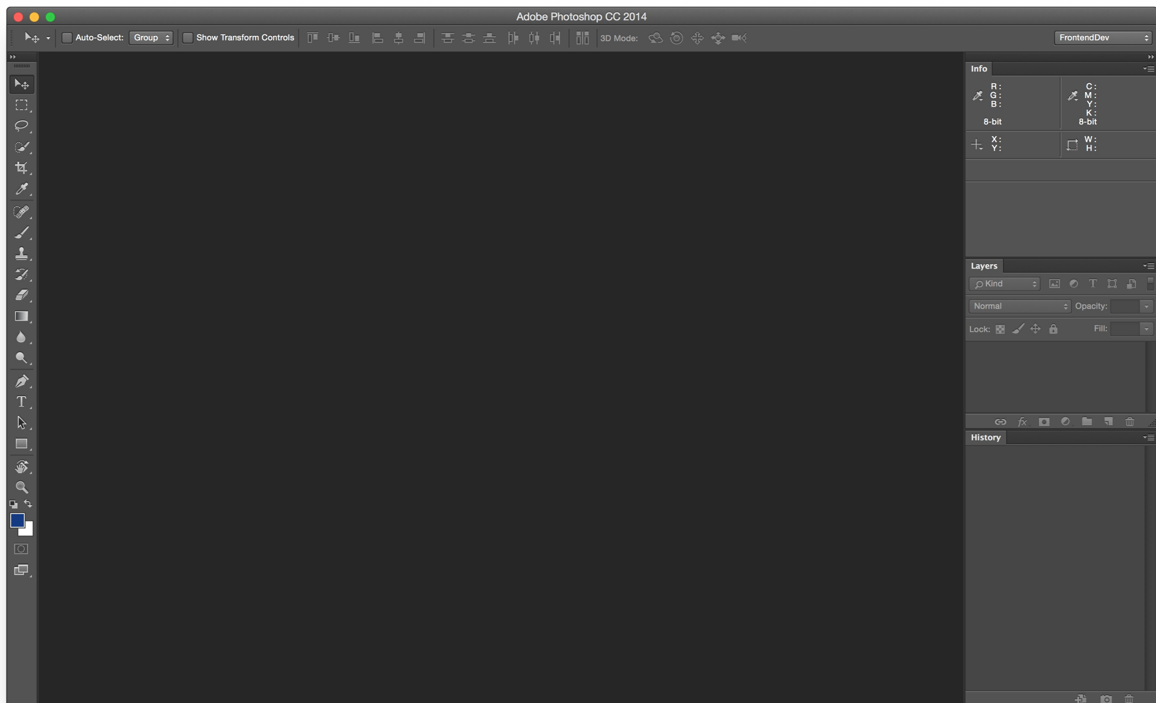
<img src="../img/avatar.jpg" alt="desc">

<style type="text/css" media="screen">
  background-image: url(../../img/sprite.png);
  background-position: 0 0;
</style>
```

工具, 面板, 视图

[TOC]

在全局设置下将单位修改为像素，因其在 CSS 中运用最广 (**Preference -> Units & Rulers -> Units**) 。设置工作区布局为切图及图片编辑做准备（所需窗口为信息窗口，图层窗口以及历史记录窗口）。



打开『信息窗口』、『图层窗口』、『历史纪录窗口』、『工具面板』、『选项』面板，可以通过 Window -> Workspace -> New WorkSpaces 保存工作区，以便下次打开。

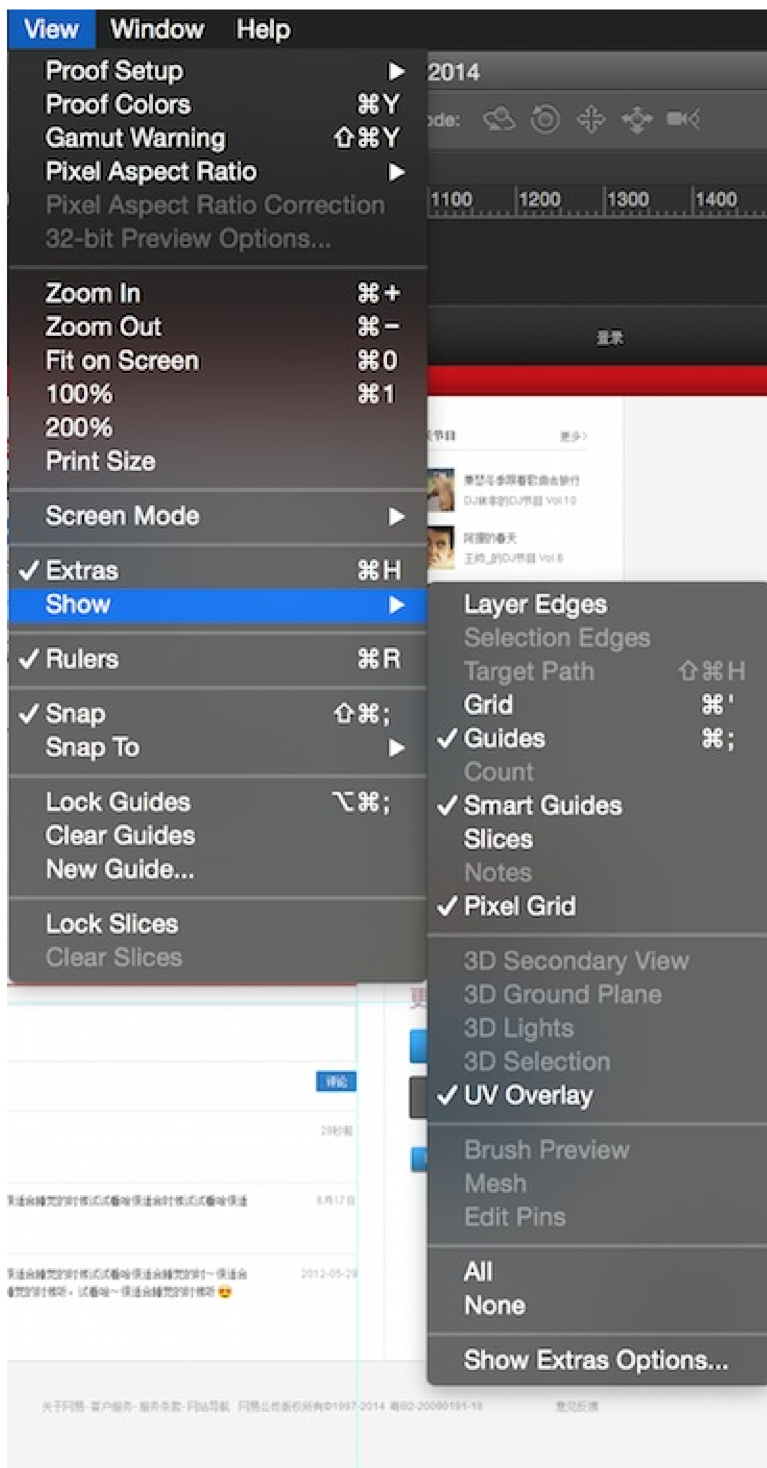
切图常用工具

工具名	示意图	注释
移动工具		
矩形选框工具		需将自动选择调至选择图层
魔棒工具		（容差 Tolerance 越小选择的范围就越小）消除锯齿可以让选择区域光滑
剪裁工具		
切片工具		在裁剪工具分支下
缩放工具		Ctrl+加号 与 Ctrl+减号 或者 alt+鼠标滑轮
取色器		

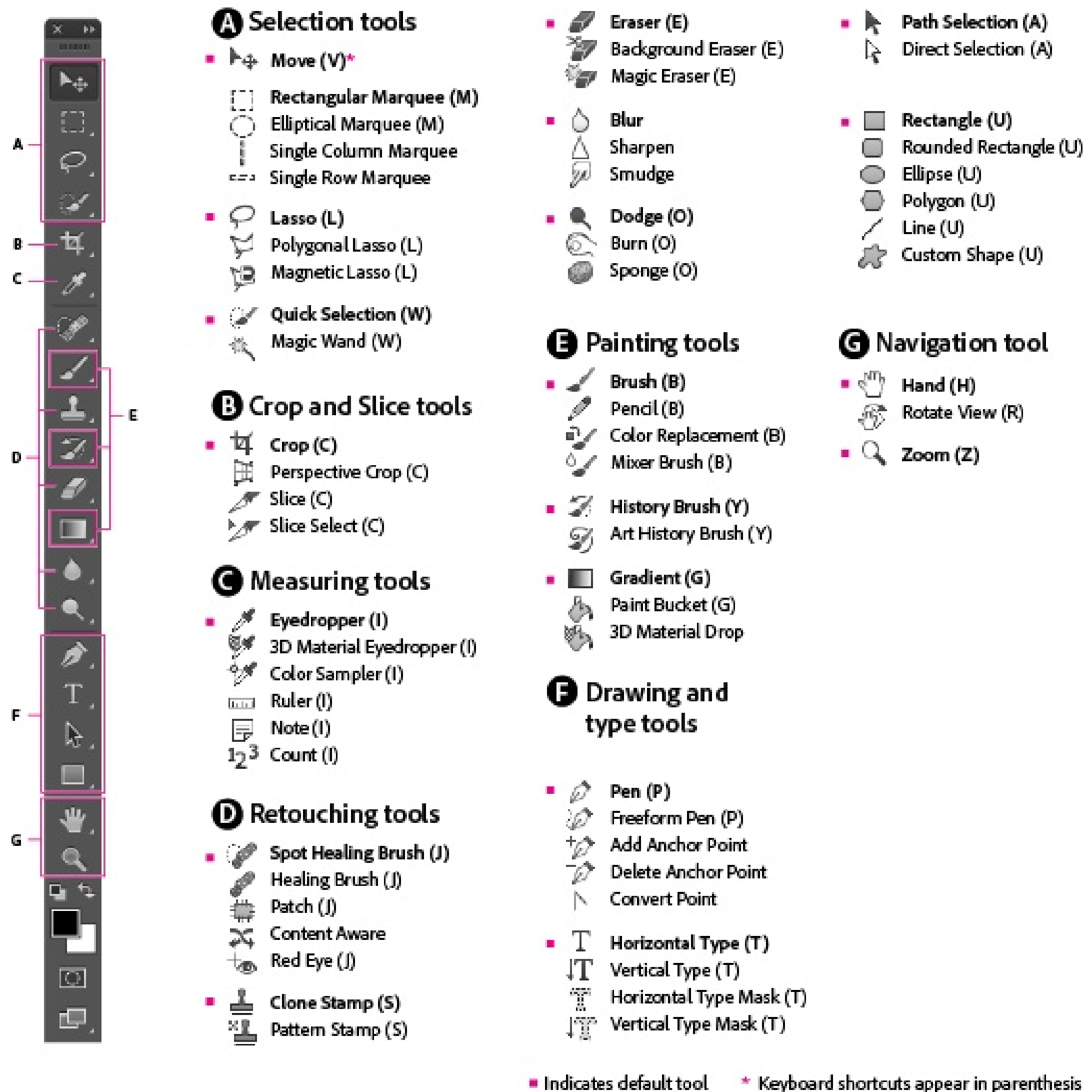
图层（单层元素）与组（类似于文件夹）的区别。

辅助视图，在视图菜单下启动

- 对齐，会启动靠近吸附功能
- 标尺，Command + R
- 参考线，Command + ;



NOTE: 所有工具及快捷键如下。



常用快捷键

放大缩小画布

- 放大到100%大小：Ctrl+数字1
- 放大：ctrl + 加号
- 缩小：ctrl + 减号
- alt + 滚轮

合并图层

- 合并图层：ctrl+E
- 合并可见图层：ctrl+shift+E

Table of Contents *generated with DocToc*

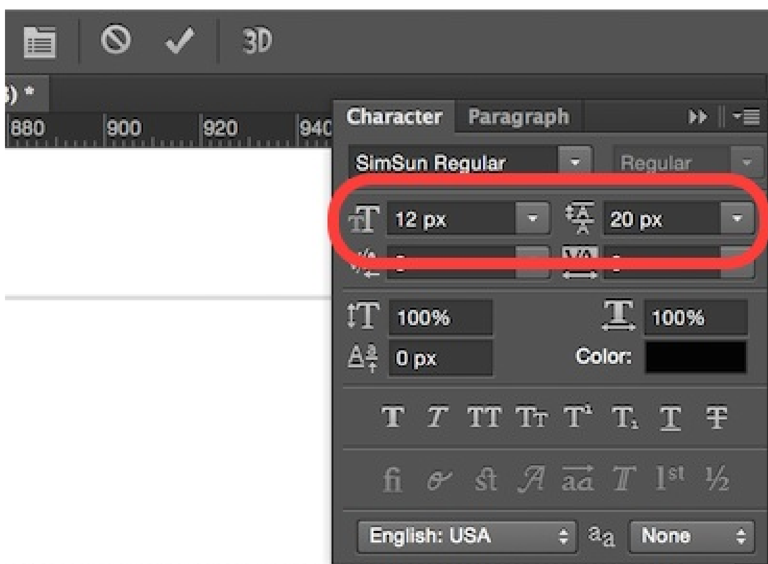
测量及取色

所有能接受数字的属性都需要测量并尽可能百分百的还原设计稿。

测量

- 宽度，高度 (width, height)
- 内外边距 (padding, margin)
- 边框 (border)
- 定位 (position)
- 文字大小 (font-size)
- 行高 (line-height)，其为第一行的底端到第二行的底端。
- 背景位置 (background-position)

NOTE: 测量时尽可能放大画布以减少误差。量取文字是为了减少误差尽量选取尺寸大的文字进行测量。



选框工具的多用途，增（Shift）减（Alt）以及交叉选择（Shift + Alt）。左右（或上下）使用分离选框选择可以得到整两个分离边框的距离总值。

取色

所有能接受颜色的属性都需要取色。

- 边框色
- 背景色
- 文字色

NOTE：使用魔棒工具可以迅速识别背景色是否没线性渐变的方法。Mac OS X 推荐使用 **Sip** 可在 Mac App Store 免费下载。 NOTE+：可以使用魔棒工具来判断颜色是否为渐变。

Table of Contents *generated with DocToc*

切图

- 内容性图片 指的是图片在页面是作为内容存在，如页面中的海报。
- 修饰性图片 指的是图片在页面中起修饰作用，如页面中的背景和图标。

修饰性图标和内容性图片需要（在 HTML 的 `<img>` 之中，只需站位不需切图）切出。切出的内容性图片应保存为 `*.jpg` 格式，而修饰性图片因保存为 `png24`（IE6 不支持半透明，Alpha 透明）或 `png8` 它们均支持全透明。

隐藏文字，方法一，直接在图层中隐藏文字图层。方法二（两种，分别应对于纯色和有背景需要隐藏文本的情况）如下图所示，使用自由变换。



PNG24切图方法

- 移动工具选中所需图层（`Ctrl` 多选）
- 右键合并图层（`Ctrl` + `E`）
- 复制到新图层

PNG8带背景切图方法

- 合并可见图层（`Shift` + `Ctrl` + `E`）
- 矩形选框选择内容
- 魔棒工具去除多余部分（`Alt` + 选取）

可平铺背景的切图方法

- 用矩形选择一个区域
- 复制至新图层

NOTE: X 轴平铺需要占满图片的宽，Y 轴平铺需要占满图片的高。

切片工具（大图化小的方法，将一大图分为多小图）

- 拉参考线
- 选择切片工具
- 点击“基于参考线的切片”按钮
- 选择切片选择工具
- 保存于新图层

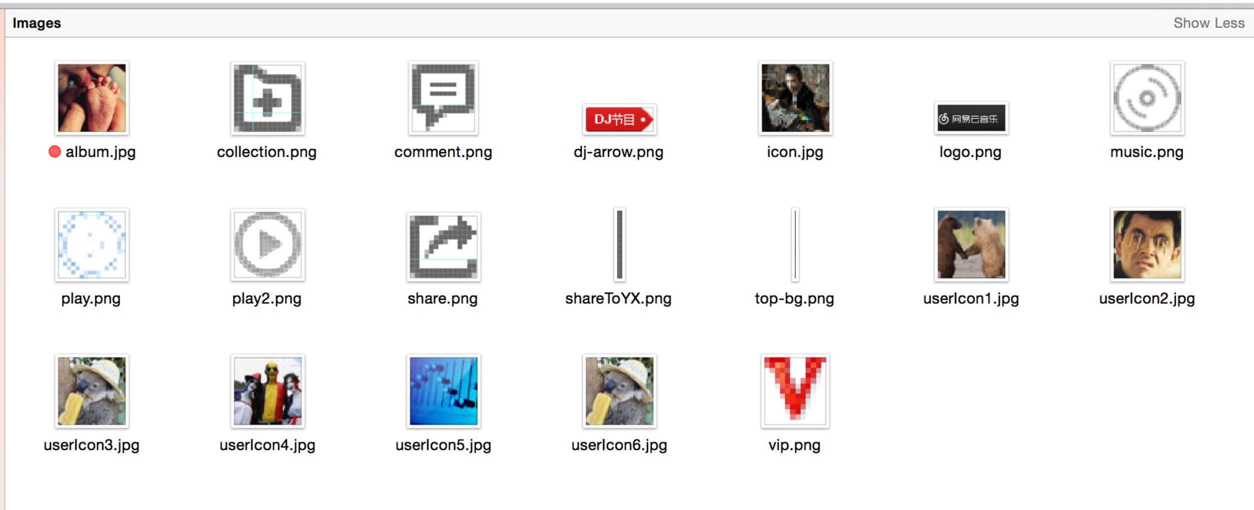
如何开始切图

第一步：去掉所有文字，只留背景

打开视觉稿后，用上面说过的方法去掉所有的文字，只留背景和图片（记得备份一下PSD文件）。

第二步：保存图片

将去掉文字的图片保存起来。一般情况下内容性图片保存为jpg格式，图标类型的保存为png格式。



第三步：构思页面的具体实现

划分页面的结构，具体的实现方式。

第四步：一边编写**HTML**代码，一边测量、取色

根据构思号的页面结构，开始编写HTML代码，并且开始进行测量和取色。

Table of Contents *generated with DocToc*

图片保存

将需要的内容保存在独立的文件里便于之后的导出。（存储于 Web 所用格式 `Alt` + `Shift` + `Ctrl` + `S`）

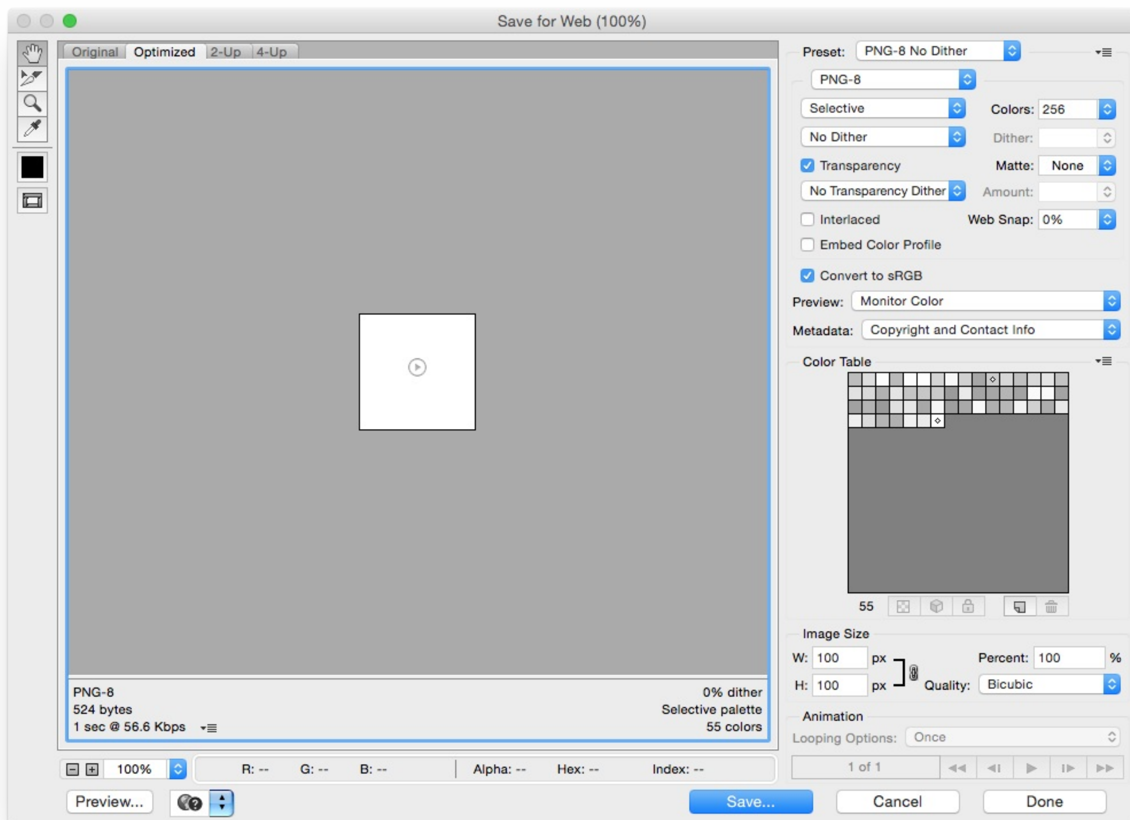
如需保存独立图层则要把需要的图层拖到新建的透明背景的图层，或在图层上右键复制（Duplicate）图层选择地址为新文件即可。

图片与背景合并的切图方法如下

保存格式的选择

保存类型一：色彩丰富切无透明要求时保存为 `JPG` 格式并选用时候的品质（通常使用品质 80 ）。

保存类型二：图片色彩不丰富，不伦透明与否一律保存为 `PNG8` 格式（256颜色，需特殊设置如下图，需设置 `杂边：无` `无仿色` ）。



保存类型三：图片有半透明（Alpha 透明）的要求，保存为 `PNG24` 格式（不对图片进行压缩）。

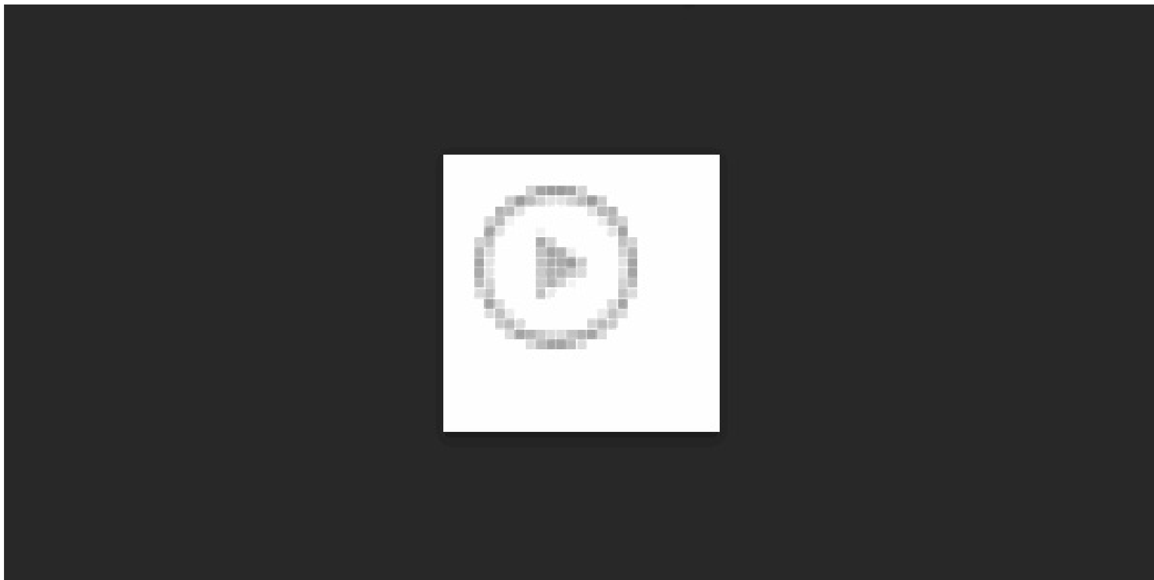
保存类型四：保留 PSD 源文件，以备不时之需。

如何保存

一般使用『存储于 Web 所用格式』菜单（Alt + Shift + Ctrl + S）保存

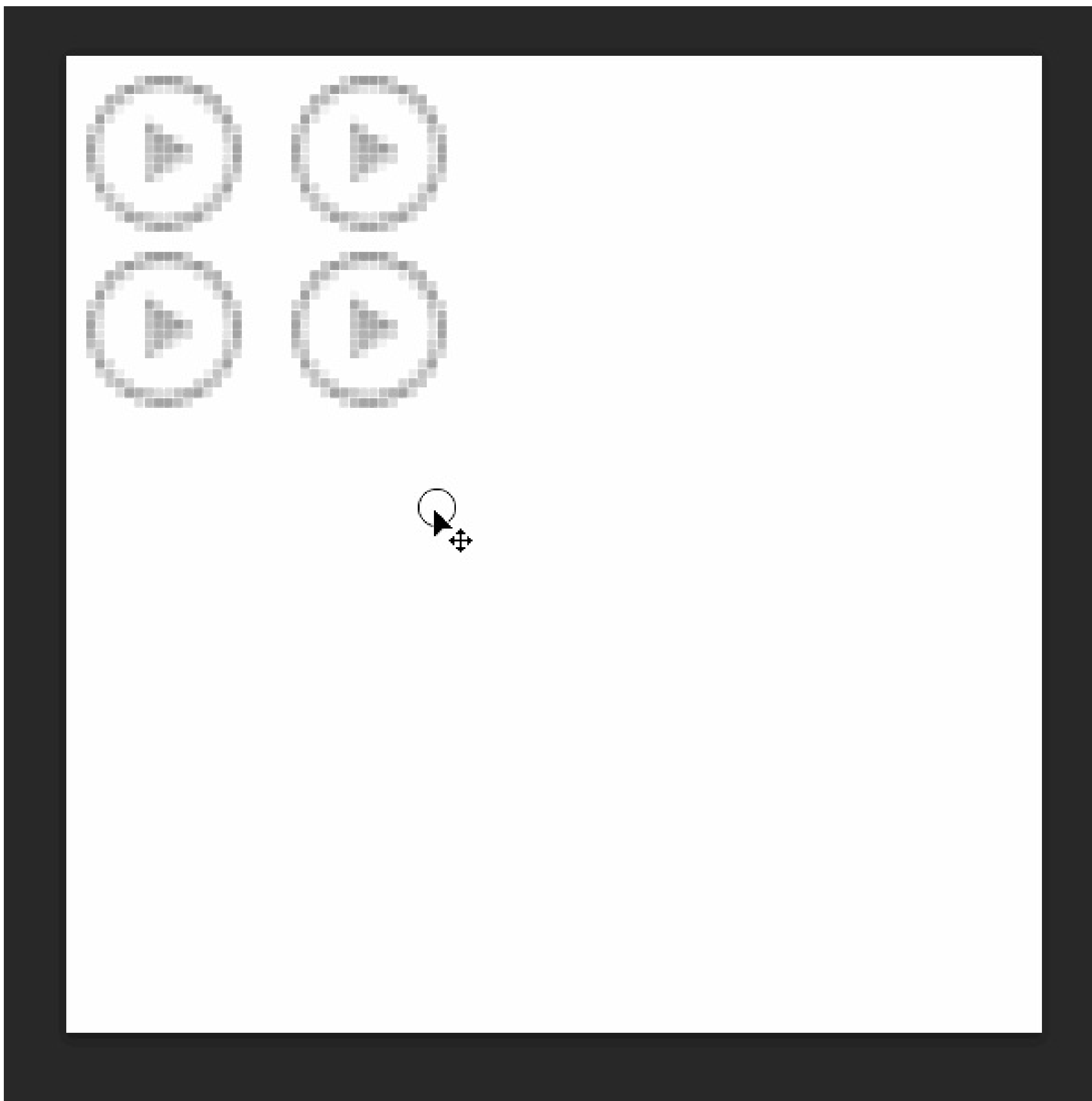
图片修改与维护

维护与修改之一：更改画布大小以便增加新素材。



维护与修改之二：移动图标分两种，独立图层（移动工具拖动），于非独立图层（选取工具选中分离后移动工具拖动）。

维护与修改之三：裁剪画布的方法有两种，(1)用选取工具选取后图片裁取，(2)直接用裁剪工具裁剪画布。



注意事项：为了可维护性的考虑因适当的留出适当的空白区域以便修改所用和提高容错性。`PNG8` 需更改图片颜色模式为 RGB 颜色（默认为索引颜色模式，颜色信息会被丢失）。

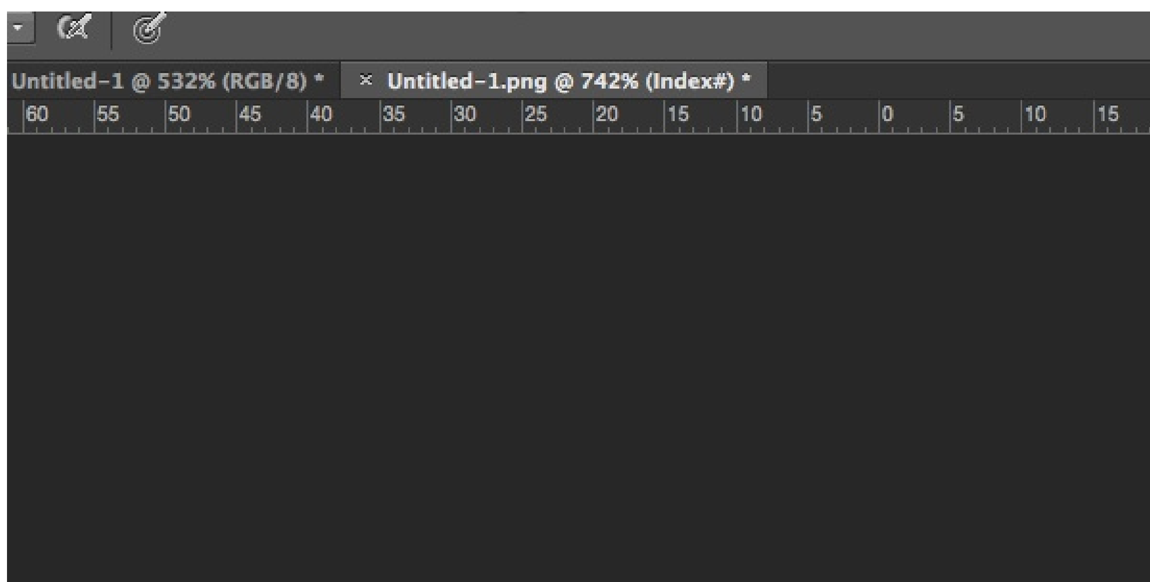

Untitled-1 @ 532% (RGB/8) *
Untitled-1.png @ 742% (Index#) *
60 55 50 45 40 35 30 25 20 15 10 5 0 5 10 15

Table of Contents *generated with [DocToc](https://github.com/thlorenz/doctoc)*

图片优化与合并

在 HTML 中使用背景图片的方法如下：

```
<button type="button" class="btn-default">Click Me</button>

<style type="text/css" media="screen">
  .btn-default {
  background: url(image/btn.png) no-repeat 0 0;
}
.btn-default-alt {
background: url(image/sprite.png) no-repeat 0 -50px;
}
</style>
```

图片的合并就如同上面提到的切图后保存的过程。拼好的图称之为 **Sprite** 它能减少网络请求次数提高速度。图片压缩工具分为无损（ImageOptim 等工具，也可结合 Grunt 自动化构建工具一同使用）与有损压缩工具（TinyPng）。

图片合并

图片合并建议方案：

- 同个模块的图片合并
- 大小相近的图片合并
- 色彩相近的图片合并
- 以上3种合并混合

合并的图片可以以横向或纵向的排列，分类可将同属于一个模块（功能模块），大小相近（充分利用画布空间），颜色相近（减少文件大小）。

图片的兼容

IE6 不支持 PNG24 半透明所以需要保存两份（sprite.png - png24 和 sprite-ie.png - 8）。在使用 CSS3 是让高级浏览器使用 CSS3 低级浏览器则使用切图。优雅降级指的是让低级浏览器不显示高级浏览器中的界面细节。

Table of Contents *generated with DocToc*

开发及调试工具

- 文本编辑器或 IDE (集成开发环境)
- Google Chrome, Firefox Firebug, Safari Developer Tool

NOTE: Google Chrome DevTools Doc

Table of Contents *generated with DocToc*

Sublime 编辑器

介绍

Sublime Text是一款性感的编辑器，具有优雅，快速，插件多等优势，不失为前端开发者的轻量高效编辑器。

安装

Windows/OS X

官方站点下载安装即可。

Ubuntu

可参见 Ubuntu 下使用 Sublime Text 并解决中文输入问题，用apt-get安装，解决任务栏，中文输入等多个问题。

Sublime 快捷键

`command/control` + `P` 进入查找命令（Goto Anything）,此时有三种选择：

- `:` 输入行数找到对应行（control + G）
- `@` 找到特定函数（command/control + R）
- `#` 找到对应变量与块

推荐插件

1.Package Control

以后的插件安装基本都靠他了，安装方法可以去Package Control查看，注意Sublime Text的版本问题。

2.Emmet

前端神器，相信搞前端的没有不用的

下面插件在**Edit** 或者 **Tools** 里面看到插件功能：

3.JQuery

写JQuery怎么能不用他来增强你的提示？

4.FileHeader

自动创建文件开头模板，并且会根据最后的保存时间修改更新时间

5.Pretty Json

写json不格式话怎么行？

6.CSS Format

css格式化

7.ConvertToUTF8

GBK编码兼容

用户自定义代码

Preferences - Settings - User 里面加入，全部的设置均为 `JSON` 文本。

```
"translate_tabs_to_spaces": true,
"tab_size": 2,
```

把Tab对齐转化为空格对齐，tab_size 控制转化比例。

```
"trim_trailing_white_space_on_save": true,
```

自动移除行尾多余空格。

```
"ensure_newline_at_eof_on_save": true,
```

自动在文件末尾加入一个空行，git 用户相信知道是干嘛的。

```
"save_on_focus_lost": true,
```

窗口失去焦后立即保存文件。

```
 "bold_folder_labels": true,
```

侧栏文件夹加粗。

Atom 文本编辑器

本文即为在 **Atom** 下编写完成，在写作过程中让我对这个崭新的 1.0 版本文本编辑器 有了更多的了解。 在阅读本文时注意快捷键于后面英文单词的对应可帮助记忆，在使用中忘记的快捷键以可以通过使用 查询面板（后面会提到）进行查询。 如果你在使用过程中发现了异常和错误可以到 **Atom** 所在的 GitHub 仓库提交问题报告。同一款编辑器一同成长，愿力量与你同在！ 下面的快捷键均为 **Mac OS X** 默认设置。如你用的是 Windows 或者是 Linux，可能需要尝试将 所有提到的 cmd 改为 `ctrl`。

基础中的基础

开始之前先把下面这条快捷键记住。`cmd+shift+P` 它会打开类似 Alfred 的快捷功能选择窗口， 如果你从来没有听过 Alfred（此为 Mac OS X 特有应用） 那你应该赶紧去所搜引擎中找找了。

保存时间

快捷键	描述
`cmd-shift-S`	可以另存为 "Save As"
`cmd-alt-S`	可以保存全部的 "Save All".

打开文件与目录

如果在命令行环境中可以使用下面的方法一次打开多个目录。

```
# 打开目录
atom ./hopes ./dreams

# 获得帮助
atom -h
```

快捷键	描述
`cmd-O`	打开文件
`cmd-shift-O`	添加目录至当前编辑器窗口

`cmd-P` 可以打开 Fuzzy Finder 进行模糊搜索，默认可所搜区域为项目内所有文件。下面的命令可以对模糊所搜做一些限制，`cmd-B` 只所搜已打开的文件（存在与 Buffer 中的文件）。 `cmd-shift-B`

1.0 版本中在编辑器中添加的新文件无法使用 Fuzzy Finder（模糊寻找） 找到，重启后则可以解决。

边栏（树目录）

快捷键	描述
`cmd-\`	显示或隐藏边栏
`ctrl-0`	聚焦边栏，聚焦后可以操作树目录中的文件

在聚焦后可以通过 `a` 来增加（add），`m` 来移动（move），`d` 来复制（duplicate）或者 `delete` 来删除（此处为键盘删除键）。这里的操作并没有自动路径补全功能，之后可能需要插件支持。

开始使用

Atom 中几乎所有的功能都是以插件的形式存在的。所有如何安装插件则就是我们第一件要做的事。除了图形界面安装的方法外，随 **Atom** 还安装了插件管理器叫做 `apm` 。通过它也可以直接安装 和更新插件。简单说主题也是插件，所以安装主题与安装插件的步骤类似。

下面的操作均需要联网

```
# 安装插件
apm install <package_name>

# 安装指定版本的插件
apm install <package_name>@<package_version>

# 查询插件
apm search <package_name>

# 查询插件详情
apm view <package_name>
```

移动光标

Atom 的移动快方法同 **Emacs** 一致。在熟悉使用 **Atom** 后也很容易的转移至 **Emacs** 的环境下熟练工作。

单个字符的移动，效果于方向键一致。

快捷键	描述
`ctrl-P`	上移（Previous）
`ctrl-N`	下移（Next）
`ctrl-B`	后移（Back）
`ctrl-F`	前移（Forward）

在单个字符移动基础上，可以延展至更大范围的移动。例如，单词，整行。

快捷键	描述
`alt-B`	向后以词为单位移动（英文），中文则以英文标点为间隔
`alt-F`	向前以词为单位移动（英文），中文则以英文标点为间隔
`ctrl-E`	移动至行末（End）
`ctrl-A`	移动至此行首字符（Ahead)
`ctrl-A` （敲击两次）	移动至此行行首（包括空格）
`cmd-up`	移动至文件最顶
`cmd-down`	移动至文件最低

`ctrl-G` 加数字可移动至目标行，使用 `row:column` 可以定位行数和列数， 使用这个方法在查找错误时变得十分方便。

`cmd-R` 可以在当前文件中（Buffer）按照符号来搜索，符号关键字指的是函数名（代码中） 或标题（文档中）。

选择

选择是在移动的基础上加入 `shift` 既可完成。特别的几种选择方法如下。

快捷键	描述
`cmd-L`	选取整行
`ctrl-shift-W`	选取当前单词（英文），中文则为整行

编辑与删除

Atom 如同其他的常用的文本编辑器一样可以直接编辑文字，并不存在特殊的模式。但了解下面的 编辑技巧可以让你使用 **Atom** 更得心应手。

编辑操作

快捷键	描述
`ctrl-T`	交换光标两边的字符（Transpose）
`ctrl-J`	将下一行同当前行合并（Join）
`ctrl-cmd-up`	向上冒泡当前行
`ctrl-cmd-down`	向下冒泡当前行
`cmd-shift-D`	复制当前行（Duplicate）
`cmd-K, cmd-U`	转换选中字符至全大写
`cmd-K, cmd-L`	转换选中字符至全小

删除操作

快捷键	描述
`ctrl-shift-K`	删除当前（Cut）
`cmd-delete`	删除此行光标后全部字符
`cmd-backspace`	删除至当前行首
`ctrl-K`	切帖至行末（Cut）
`alt-H`	删除前一个字符
`alt-D`	删除后一个字符

多个光标及选择

同 **Sublime Text** 相同，**Atom** 也同样有多光标的实现。 按住cmd可以在文本中使用进行区域性选择。

快捷键	描述
`cmd-click`	在点击处增加新光标
`cmd-shift-L`	将选择区域转换为多光标
`ctrl-shift-up`	在上一行增加新光标
`ctrl-shift-down`	在下一行增加新光标
`cmd-D`	选择下一个于当前被选字符相同的字符并添加新光标
`cmd-ctrl-G`	选择全部于当前选中字符相同的字符并添加光标

括号

编程中最常打交道和需要跳出的莫属于括号和引号了。**Atom** 对于括号有很好的处理办法，各种括号在光标内移动都会被自动高亮（引号和 HTML 中的标签也会被高亮和自动补全）。选中内容后使用括号可以自动将选中内容包含在括号或引号内。

快捷键	描述
`ctrl-M`	跳至最近的一个括号的起始位置
`ctrl-cmd-M`	选中括号内的所有内容
`alt-cmd-.`	关闭最近的一个 XML/HTML 标签

搜索与替换

快捷键	描述
`cmd-F`	当前文本中搜索
`cmd-shift-F`	搜索整个项目
`cmd-G`	找到下一个匹配的搜索结果
`cmd-G-shift`	找到上一个匹配的搜索结果

在项目搜索中可以使用 wildcard 和指定目标的搜索路径。

代码片段（**Snippets**）

代码片段让你在写代码时有飞一般的感觉，代码片段会将预先设置好的代码片段替换在当前文本中，并且设置焦点并用 `tab` 聚焦下一个焦点，或 `shift` + `tab` 聚焦上一个焦点。

所有的代码片都存储在下面的目录中 `~/.atom/snippets.cson` ，你可以通过 Open Your Snippets Menu 打开此文件。

快捷键	描述
`alt-shift-S`	显示当前文件类型下的全部代码片段

当然制作代码片也有一个代码片，它就是 `snip` 。

制作代码片段

下面是一个简单的代码片样例。

```
'.source.js':
  'console.log':
    'prefix': 'log'
    'body': 'console.log(${1: "crash"});$2'
```

- `.source.js` 为代码片可用的文件类型范围
- `console.log` 为代码片内容描述
- `prefix` 为代码片调用字符
- `body` 代码片主体内容
- `${1:'crash'}` 用于定义焦点，顺序及其默认值

多行代码代码片

```
'.source.js':
  'if, else if, else':
    'prefix': 'ieie'
    'body': """
      if (${1:true}) {
        $2
      } else if (${3:false}) {
        $4
      } else {
        $5
      }
    """
```

代码折叠

可以点击代码行号边的箭头折叠当前层级的代码。

快捷键	描述
`alt-cmd-[`	折叠当前层级
`alt-cmd-]`	展开当前层级
`alt-cmd-shift-{`	折叠全部层级
`alt-cmd-shift-}`	展开全部层级
`cmd-K, cmd-N` （层级数）	根据层级级别折叠

多窗口模式

任意一个窗口都可以无需的四面分割，分割的部分则依然使用标签来表示。

快捷键	描述
`cmd-k arrow`	根据方向指定分割窗口
`cmd-K, cmd-arrow`	聚焦指定方向的窗口

解码（**Encoding**）

Atom 支持多种解码格式（包括中文 GBK 的支持），也可自动识别解码方式（不能识别时则默认为 UTF-8）。当然你也可以使用这种方法将多种文本在多种解码格式直接转换。

快捷键	描述
`ctrl-shift-U`	切换解码方式

书签

在 **Atom** 添加书签就如同你看书的时添加书签一样，它使你在书写代码时可以自如的跳转到你需要的 位置。

快捷键	描述
`F2-cmd`	可以在当前行切换标记书签
`F2`	跳转至下一个书签
`F2-shift`	跳转至上一个书签
`F2-ctrl`	查看全部的书签
`F2-cmd-shift`	清除全部标签

扩展插件

下面列出了笔者在日常 Web 开发中所易用的插件，这些插件满足了超过百分之九十笔者的开发需求。下面的这些插件均可以在官方插件管理器中进行下载和安装。

```
[
    {
        "name": "advanced-new-file",
        "version": "0.4.3"
    },
    {
        "name": "advanced-open-file",
        "version": "0.8.2"
    },
    {
        "name": "atom-beautify",
        "version": "0.28.8"
    },
    {
        "name": "AtomicChar",
        "version": "0.3.8"
    },
    {
        "name": "autocomplete-paths",
        "version": "1.0.2"
    },
    {
        "name": "docblockr",
        "version": "0.7.3"
    },
    {
```

```
            "name": "ex-mode",
            "version": "0.7.0"
        },
        {
            "name": "file-icons",
            "version": "1.6.2"
        },
        {
            "name": "language-jade",
            "version": "0.6.2"
        },
        {
            "name": "linter",
            "version": "1.3.0"
        },
        {
            "name": "linter-jshint",
            "version": "1.1.4"
        },
        {
            "name": "markdown-toc",
            "version": "0.3.0"
        },
        {
            "name": "merge-conflicts",
            "version": "1.3.5"
        },
        {
            "name": "minimap",
            "version": "4.12.2"
        },
        {
            "name": "minimap-linter",
            "version": "1.0.0"
        },
        {
            "name": "open-in-browser",
            "version": "0.4.6"
        },
        {
```

```
        "name": "pigments",
        "version": "0.9.3"
    },
    {
        "name": "sort-lines",
        "version": "0.11.0"
    },
    {
        "name": "sync-settings",
        "version": "0.6.0"
    },
    {
        "name": "tab-switcher",
        "version": "1.2.1"
    },
    {
        "name": "theme-switcher",
        "version": "1.1.0"
    },
    {
        "name": "vim-mode",
        "version": "0.57.0"
    },
    {
        "name": "atom-dark-syntax",
        "version": "0.27.0",
        "theme": "syntax"
    },
    {
        "name": "atom-dark-ui",
        "version": "0.49.0",
        "theme": "ui"
    },
    {
        "name": "atom-light-syntax",
        "version": "0.28.0",
        "theme": "syntax"
    },
    {
        "name": "atom-light-ui",
```

```
        "version": "0.41.0",
        "theme": "ui"
    },
    {
        "name": "base16-tomorrow-dark-theme",
        "version": "0.26.0",
        "theme": "syntax"
    },
    {
        "name": "base16-tomorrow-light-theme",
        "version": "0.9.0",
        "theme": "syntax"
    },
    {
        "name": "one-dark-ui",
        "version": "1.0.2",
        "theme": "ui"
    },
    {
        "name": "one-dark-syntax",
        "version": "1.1.0",
        "theme": "syntax"
    },
    {
        "name": "one-light-syntax",
        "version": "1.1.0",
        "theme": "syntax"
    },
    {
        "name": "one-light-ui",
        "version": "1.0.2",
        "theme": "ui"
    },
    {
        "name": "solarized-dark-syntax",
        "version": "0.38.1",
        "theme": "syntax"
    },
    {
        "name": "solarized-light-syntax",
```

```
            "version": "0.22.1",
            "theme": "syntax"
        },
        {
            "name": "about",
            "version": "1.0.1"
        },
        {
            "name": "archive-view",
            "version": "0.58.0"
        },
        {
            "name": "autocomplete-atom-api",
            "version": "0.9.2"
        },
        {
            "name": "autocomplete-css",
            "version": "0.9.0"
        },
        {
            "name": "autocomplete-html",
            "version": "0.7.2"
        },
        {
            "name": "autocomplete-plus",
            "version": "2.19.0"
        },
        {
            "name": "autocomplete-snippets",
            "version": "1.7.1"
        },
        {
            "name": "autoflow",
            "version": "0.25.0"
        },
        {
            "name": "autosave",
            "version": "0.22.0"
        },
        {
```

```
            "name": "background-tips",
            "version": "0.25.0"
        },
        {
            "name": "bookmarks",
            "version": "0.35.0"
        },
        {
            "name": "bracket-matcher",
            "version": "0.76.0"
        },
        {
            "name": "command-palette",
            "version": "0.36.0"
        },
        {
            "name": "deprecation-cop",
            "version": "0.53.1"
        },
        {
            "name": "dev-live-reload",
            "version": "0.46.0"
        },
        {
            "name": "encoding-selector",
            "version": "0.21.0"
        },
        {
            "name": "exception-reporting",
            "version": "0.36.0"
        },
        {
            "name": "find-and-replace",
            "version": "0.175.0"
        },
        {
            "name": "fuzzy-finder",
            "version": "0.87.0"
        },
        {
```

```
            "name": "git-diff",
            "version": "0.55.0"
        },
        {
            "name": "go-to-line",
            "version": "0.30.0"
        },
        {
            "name": "grammar-selector",
            "version": "0.47.0"
        },
        {
            "name": "image-view",
            "version": "0.54.0"
        },
        {
            "name": "incompatible-packages",
            "version": "0.24.1"
        },
        {
            "name": "keybinding-resolver",
            "version": "0.33.0"
        },
        {
            "name": "link",
            "version": "0.30.0"
        },
        {
            "name": "markdown-preview",
            "version": "0.150.0"
        },
        {
            "name": "metrics",
            "version": "0.51.0"
        },
        {
            "name": "notifications",
            "version": "0.57.0"
        },
        {
```

```
            "name": "open-on-github",
            "version": "0.37.0"
        },
        {
            "name": "package-generator",
            "version": "0.40.0"
        },
        {
            "name": "release-notes",
            "version": "0.53.0"
        },
        {
            "name": "settings-view",
            "version": "0.211.2"
        },
        {
            "name": "snippets",
            "version": "0.95.0"
        },
        {
            "name": "spell-check",
            "version": "0.59.0"
        },
        {
            "name": "status-bar",
            "version": "0.75.1"
        },
        {
            "name": "styleguide",
            "version": "0.44.0"
        },
        {
            "name": "symbols-view",
            "version": "0.100.0"
        },
        {
            "name": "tabs",
            "version": "0.82.0"
        },
        {
```

```
            "name": "timecop",
            "version": "0.31.0"
        },
        {
            "name": "tree-view",
            "version": "0.180.0"
        },
        {
            "name": "update-package-dependencies",
            "version": "0.10.0"
        },
        {
            "name": "whitespace",
            "version": "0.30.0"
        },
        {
            "name": "wrap-guide",
            "version": "0.35.0"
        },
        {
            "name": "language-c",
            "version": "0.45.0"
        },
        {
            "name": "language-clojure",
            "version": "0.16.0"
        },
        {
            "name": "language-coffee-script",
            "version": "0.41.0"
        },
        {
            "name": "language-csharp",
            "version": "0.6.0"
        },
        {
            "name": "language-css",
            "version": "0.32.1"
        },
        {
```

```
            "name": "language-gfm",
            "version": "0.80.0"
        },
        {
            "name": "language-git",
            "version": "0.10.0"
        },
        {
            "name": "language-go",
            "version": "0.31.0"
        },
        {
            "name": "language-html",
            "version": "0.40.0"
        },
        {
            "name": "language-hyperlink",
            "version": "0.14.0"
        },
        {
            "name": "language-java",
            "version": "0.15.0"
        },
        {
            "name": "language-javascript",
            "version": "0.85.0"
        },
        {
            "name": "language-json",
            "version": "0.15.0"
        },
        {
            "name": "language-less",
            "version": "0.28.1"
        },
        {
            "name": "language-make",
            "version": "0.14.0"
        },
        {
```

```
            "name": "language-mustache",
            "version": "0.12.0"
        },
        {
            "name": "language-objective-c",
            "version": "0.15.0"
        },
        {
            "name": "language-perl",
            "version": "0.28.0"
        },
        {
            "name": "language-php",
            "version": "0.27.0"
        },
        {
            "name": "language-property-list",
            "version": "0.8.0"
        },
        {
            "name": "language-python",
            "version": "0.38.0"
        },
        {
            "name": "language-ruby",
            "version": "0.57.0"
        },
        {
            "name": "language-ruby-on-rails",
            "version": "0.22.0"
        },
        {
            "name": "language-sass",
            "version": "0.40.0"
        },
        {
            "name": "language-shellscript",
            "version": "0.15.0"
        },
        {
```

```
            "name": "language-source",
            "version": "0.9.0"
        },
        {
            "name": "language-sql",
            "version": "0.17.0"
        },
        {
            "name": "language-text",
            "version": "0.7.0"
        },
        {
            "name": "language-todo",
            "version": "0.25.0"
        },
        {
            "name": "language-toml",
            "version": "0.16.0"
        },
        {
            "name": "language-xml",
            "version": "0.30.0"
        },
        {
            "name": "language-yaml",
            "version": "0.22.0"
        }
    ]
```

Table of Contents *generated with DocToc*

HTML

HTML 历史

HTML (Hyper Text Markup Language)，用于标记页面中的内容。

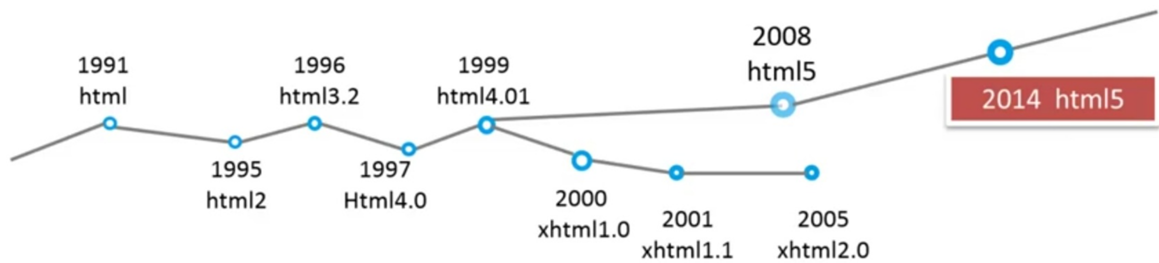


Table of Contents *generated with DocToc*

HTML 简介

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name= “keywords” content= “音乐...” />
    <meta name= “description” content= “网易...” />
    <title>网易云音乐 听见好时光</title>
      ...
  </head>
  <body>
    <iframe>...</iframe>
    <div class="g-btmbar">...</div>
  </body>
</html>
```

文档声明

文档头部

文档主体

注意事项：

- `<!DOCTYPE html>` 必须首行定格
- `<title>` 为文档标题
- `<meta charset="utf-8">` 文档解码格式
- `<meta name="keywords" content="...">` 和 `<meta name="description" content="...">` 提供给搜索引擎使用
- `<meta name="viewport" content="width=device-width, initial-scale=1.0">` 移动端浏览器的宽高与缩放
- `<link>` 标签可以引入 favicon 和样式表 CSS 文件

Table of Contents *generated with DocToc*

HTML 语法

```
<!-- 登录窗口 -->
<div class="m-win">
    <form class="m-login" action="#">
        <fieldset>
            <legend>网易通行证登录</legend>
            <input type="text" value="帐号">
            <input type="text" value="密码">
            <button type="submit" class="u-btn">登录</button>
        </fieldset>
    </form>
</div>
```

< >

闭合

一个或多个属性值

嵌套

<!-- -->

书写规范：

- 小写标签和属性
- 属性值双引号
- 代码因嵌套缩进

全局属性

- id, `<div id='unique-element'></div>`，页面中唯一
- class， `<button class='btn'>Click Me</button>`，页面中可重复出现
- style，尽量避免
- title，对于元素的描述类似于 Tooltip 的效果。

Table of Contents *generated with DocToc*

HTML 标签

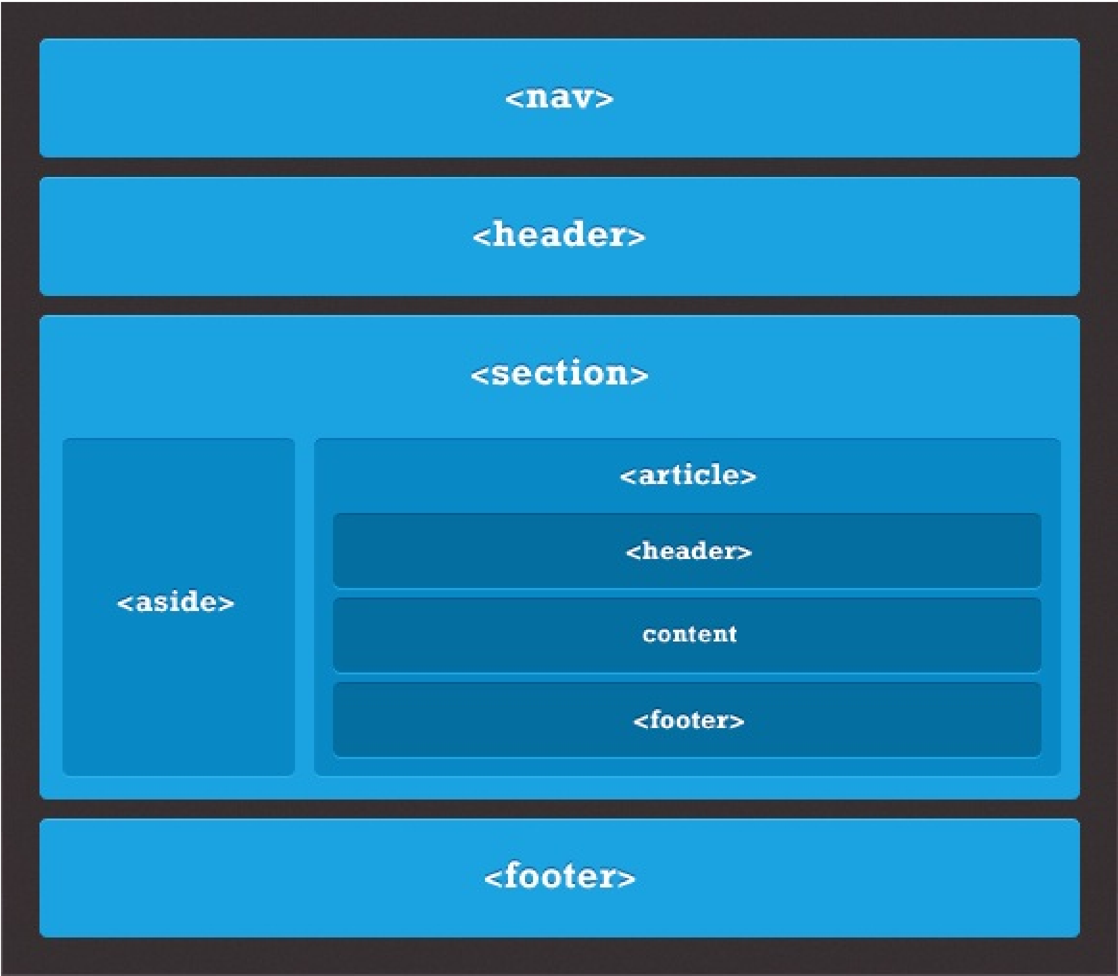
HTML5 标签集合



文档章节

`<body>` 页面内容 `<header>` 文档头部 `<nav>` 导航 `<aside>` 侧边栏 `<article>` 定义外部内容（如外部引用的文章） `<section>` 一个独立的块 `<footer>` 尾部

页面通常结构


<nav>
<header>
<section>
<aside>
<article>
<header>
content
<footer>
<footer>

文本标签

```
<!-- 默认超链接  -->
<a href="http://sample-link.com" title="Sample Link">Sample</a>
<!-- 当前窗口显示 -->
<a href="http://sample-link.com" title="Sample Link" target="_se
lf">Sample</a>
<!-- 新窗口显示 -->
<a href="http://sample-link.com" title="Sample Link" target="_bl
ank">Sample</a>
<!-- iframe 中打开链接 -->
<a href="http://sample-link.com" title="Sample Link" target="ifr
ame-name">Sample</a>
<iframe name="iframe-name" frameborder="0"></iframe>

<!-- 页面中的锚点 -->
<a href="#achor">Achor Point</a>
<section id="achor">Achor Content</section>

<!-- 邮箱及电话需系统支持 -->
<a href="mailto:sample-address@me.com" title="Email">Contact Us</
a>
<!-- 多个邮箱地址 -->
<a href="mailto:sample-address@me.com, sample-address0@me.com" t
itle="Email">Contact Us</a>
<!-- 添加抄送，主题和内容 -->
<a href="mailto:sample-address@me.com?cc=admin@me.com&subject=He
lp&body=sample-body-text" title="Email">Contact Us</a>

<!-- 电话示例 -->
<a href="tel:99999999" title="Phone">Ring Us</a>
```

组合内容标签

- `<div>`
- `<p>`
- `<ol>`
- `<ul>`
- `<dl>`
- `<pre>`

- `<blockquote>`

文档章节

`<body>` 页面内容 `<header>` 文档头部 `<nav>` 导航 `<aside>` 侧边栏 `<article>` 定义外部内容（如外部引用的文章） `<section>` 一个独立的块 `<footer>` 尾部

引用

- `<cite>` 引用作品的名字、作者的名字等
- `<q>` 引用一小段文字（大段文字引用用 `<blockquote>` ）
- `<blockquote>` 引用大块文字
- `<pre>` 保存格式化的内容（其空格、换行等格式不会丢失）

```
<pre>
  <code>
    int main(void) {
      printf('Hello, world!');
      return 0;
  }
</code>
</pre>
```

代码

`<code>` 引用代码

格式化

`<b>` 加粗 `<i>` 斜体

强调

`<em>` 斜体。着重于强调内容，会改变语义的强调 `<strong>` 粗体。着重于强调内容的重要性

换行

`<br>` 换行

列表

无序列表

```
<ul>
  <li>标题</li>
  <li>结论</li>
</ul>
```

有序列表

```
<ol>
  <li>第一</li>
  <li>第二</li>
</ol>
```

自定义列表

```
<dl>
  <dt>作者</dt>
  <dd>爱因斯坦</dd>
  <dt>作品</dt>
  <dd>《相对论》</dd>
  <dd>《时间与空间》</dd>
</dl>
```

一个 `<dt>` 可以对应多个 `<dd>`

NOTE: `<dl>` 为自定义列表，其中包含一个或多个 `<dt>` 及 一个或多个 `<dd>`，并且 `dt` 与 `dl` 列表会有缩进的效果。 `<pre>` 会保留换行和空格，通常与 `<code>` 一同使用。

```
<pre>
  <code>
    int main(void) {
      printf('Hello, world!');
      return 0;
    }
  </code>
</pre>
```

`<blockquote>` 拥有 `cite` 属性，它包含引用文本的出处，示例如下所示：

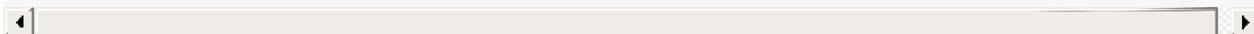
```
<blockquote cite="http://example.com/facts">
  <p>Sample Quote...</p>
</blockquote>
```

嵌入

```
<iframe src=""></iframe> 页面操作可以不影响到iframe的内容

<!--object embed通常用来嵌入外部资源 -->
<object type="application/x-shockwave-player">
  <param name="movie" value="book.pdf">
</object>

<!--视频 track可以引入字幕 autoplay可以使视频加载后自动播放，loop可以使
其循环播放 -->
<video autoplay loop controls="controls" poster="poster.jpg">
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.webm" type="video/webm">
  <source src="movie.ogg" type="video/ogg">
  <track kind="subtitles" src="video.vtt" srclang="cn" label="cn"
>
</video>
```



资源标签

图标签

`canvas` 基于像素，性能要求比较高，可用于实时数据展示。 `svg` 为矢量图形图像。

热点区域标签

`img` 中套用 `map` 以及 `area` 可以实现点击某部分图片触发一个链接，点击另一部分触发另一个链接

```
<img src="mama.jpg" width=100 height=100 usemap="#map" />
<map name="map">
    <area shap="rect" coords="0,0,50,50" href="" alt="">
    <area shap="circle" coords="75,75,25" href="" alt="">
</map>
```

表格

表格代码示例

```
<table>
  <caption>table title and/or explanatory text</caption>
  <thead>
    <tr>
      <th>header</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>data</td>
    </tr>
  </tbody>
</table>
```

使用 `colspan=val` 进行跨列，使用 `rowspan=val` 进行跨行。

表单

```
<form action="WebCreation_submit" method="get" accept-charset="utf-8">
  <fieldset>
    <legend>title or explanatory caption</legend>
    <!-- 第一种添加标签的方法 -->
    <label><input type="text/submit/hidden/button/etc" name="" value=""></label>
    <!-- 第二种添加标签的方法 -->
    <label for="input-id">Sample Label</label>
    <input type="text" id="input-id">
  </fieldset>
  <fieldset>
    <legend>title or explanatory caption</legend>
    <!-- 只读文本框 -->
    <input type="text" readonly>
    <!-- 隐藏文本框，可提交影藏数据 -->
    <input type="text" name="hidden-info" value="hiden-info-value" hidden>
  </fieldset>
  <button type="submit">Submit</button>
  <button type="reset">Reset</button>
</form>
```

使用 `fieldset` 可用于对表单进行分区

表单中的其他控件类型：

- `textarea` （文本框）
- `select` 与 `option` （下拉菜单可多选）

input 类型支持值列表

Value	Description
button	Defines a clickable button (mostly used with a JavaScript to activate a script)
checkbox	Defines a checkbox
color	Defines a color picker
date	Defines a date control (year, month and day (no time))
datetime	The input type datetime has been removed from the HTML standard. Use datetime-local instead.
datetime-local	Defines a date and time control (year, month, day, hour, minute, second, and fraction of a second (no time zone)
email	Defines a field for an e-mail address
file	Defines a file-select field and a "Browse..." button (for file uploads)
hidden	Defines a hidden input field
image	Defines an image as the submit button
month	Defines a month and year control (no time zone)
number	Defines a field for entering a number
password	Defines a password field (characters are masked)
radio	Defines a radio button
range	Defines a control for entering a number whose exact value is not important (like a slider control)
reset	Defines a reset button (resets all form values to default values)
search	Defines a text field for entering a search string
submit	Defines a submit button
tel	Defines a field for entering a telephone number
text	Default. Defines a single-line text field (default width is 20 characters)
time	Defines a control for entering a time (no time zone)
url	Defines a field for entering a URL
week	Defines a week and year control (no time zone)

语义化

语义化（Semantic Tag）是指用合适的标签标识适当的内容，它可以起到搜索引擎优化（Search Engine Optimization），提高可访问性（例如盲人使用的屏幕阅读器），与此同时还可以提高代码的可读性。简而言之也就是在正确的地方使用正确的标签

Table of Contents *generated with [DocToc](https://github.com/thlorenz/doctoc)*

实体字符

实体字符（ASCII Encoding Reference）是用来在代码中以实体代替与HTML语法相同的字符，避免浏览解析错误。它的两种表示方式，第一种为 `&` 外加实体字符名称，例如 ` ` ，第二种为 `&` 加实体字符序号，例如 ` ` 。

常用HTML字符实体（建议使用实体）：

字符	名称	实体名	实体数
"	双引号	"	"
&	&符	&	&
<	左尖括号（小于号）	<	<
>	右尖括号（大于号）	>	>
	空格		
	中文全角空格	&	

常用特殊字符实体（不建议使用实体）：

字符	名称	实体名	实体数
¥	元	¥	¥
¦	断竖线	¦	¦
©	版权	©	©
®	注册商标R	®	®
™	商标TM	™	™
·	间隔符	·	·
«	左双尖括号	«	«
»	右双尖括号	»	»
°	度	°	°
×	乘	×	×
÷	除	÷	÷
‰	千分比	‰	‰

NOTE：具体所需可在使用时查询，无需记忆。

Table of Contents *generated with DocToc*

浏览器兼容

主流浏览器都兼容 HTML5 的新标签，对于 IE8 及以下版本不认识 HTML5的新元素，可以使用 JavaScript 创建一个没用的元素来解决，例如：

```
<script>
    document.createElement("header");
</script>
```

也可以使用 shiv 来解决兼容性问题，详情可参考 HTML5 Shiv

Table of Contents *generated with DocToc*

CSS

简介

CSS (Cascading Stylesheet) 它用于设置页面的表现。CSS3 并不是一个完整的独立版本而是将不同的功能拆分成独立模块（例如，选择器模块，盒模型模块），这有利于不同功能的及时更新与发布也利于浏览器厂商的实习。



CSS 引入方法

```
// 外部样式表
<head>
  <link rel="stylesheet" type="text/css" href="style.css">
</head>

// 内部样式表
<head>
  <style type="text/css">
    p {
      margin: 10px;
    }
  </style>
</head>

// 内嵌样式(可在动态效果中同 JavaScript 一同使用)
<p style="color: red">Sample Text</p>
```

Table of Contents *generated with DocToc*

语法

```
/* 选择器 */
.m-userlist {
  /* 属性声明 */
  margin: 0 0 30px;
  /* 属性名:属性值; */
}
.m-userlist .list {
  position: relative;
  height: 100px;
  overflow: hidden;
}
```

浏览器私有属性

- Google Chrome, Safari (`-webkit`)
- Firefox (`-moz-`)
- IE (`-ms-`)
- Opera (`-o-`)

```
.pic {
  -webkit-transform: rotate(-3deg);
  -ms-transform: rotate(-3deg);
  transform: rotate(-3deg);
}
```

NOTE: 使用 http://pleeease.io/play/ ，CSS 预处理器（Sass，Less，Stylus）或编辑器插件可自动添加浏览器厂商的私有属性前缀。

属性值语法

```
margin: [ <length> | <percentage> | auto ]{1,4}
/* 基本元素：<length>, <percentage>, auto*/
/* 组合符号：[], | */
/* 数量符号：{1,4} */
```

基本元素

关键字

- auto
- solid
- bold
- ...

类型

- 基本类型
  - `<length>`
  - `<percentage>`
  - `<color>`
  - ...
- 其他类型
  - <'padding-width'>
  - <'color-stop'>
- 符号
  - `/`
  - ','
- inherit, initial

组合符号

- `<'font-size'> <'font-family'>` （` ` 两项必存，顺序毕遵）
  - 合法： `12px arial`

  - 不合法： `2em`
  - 不合法： `arial 14px`
- `<length>&&<color>` (`&&` 两项必存，顺序无碍)
  - 合法：green 2px
  - 合法：1em orange
  - 不合法：blue
- `underline || overline || line-through || blink` (`||` 至少选一，顺序无碍)
  - 合法：underline
  - 合法：overline underline
- `<color> | transparent` （ `|` 只可选一，不可共存）
  - 合法：orange
  - 合法：transparent
  - 不合法：orange transparent
- `bold [thin || <length>]` （ `[]` 分组之用，视为整体）
  - 合法：bold thin
  - 合法：bold 2em

数量符号

- `<length>` （无则表示仅可出现一次）
  - 合法：1px
  - 合法：10em
  - 不合法：1px 2px
- `<color-stop>[, <color-stop>]+` (`+` 可出现一次或多次)
  - 合法：#fff, red
  - 合法：blue, green 50%, gray
  - 不合法：red
- `inset?&&<color>` (`?` 表示可选)
  - 合法：inset orange
  - 合法：red
- `<length>{2,4}` (`{2,4}` 可出现次数和最少最多出现次数)
  - 合法：1px 2px
  - 合法：1px 2px 3px
  - 不合法: 1px
  - 不合法：1px 2px 3px 4px 5px

- `<time>[, <time>]*` （ `*` 出现 0 次或多次）
  - 合法：1s
  - 合法：1s,4ms
- `<time>#` （ `#` 出现一次或者多次，用 `,` 分隔）
  - 合法：2s, 4s
  - 不合法：1s 2s

CSS 规则示例

padding-top: <length>|<percentage>

padding-top:1px;

padding-top:1em 5%;

border-width:[<length> | thick | medium | thin]{1,4}

border-width:2px;

border-width:2px small;

box-shadow: [inset? && [<length>{2,4} && <color>?]]#|none

box-shadow:3px 3px rgb(50%, 50%, 50%), red 0 0 4px inset;

box-shadow:inset 2px 4px, 2px blue;

@规则语法

```
@import "subs.css";
@charset "utf-8";
@media print {
  /* property: value */
}
@keyframes fadein {
  /* property: value */
}
```

- `@标示符 内容;`
- `@标示符 内容{}`

@规则

常用的规则

- `@media` （用于响应式布局）

- `@keyframes` （CSS 动画的中间步骤）
- `@font-face` （引入外部字体）

其他规则（不常用）

- `@import`
- `@charset`
- `@namespace`
- `@page`
- `@supports`
- `@document`

Table of Contents *generated with DocToc*

选择器

See the Pen FEND_Selectors by Li Xinyang (@li-xinyang) on CodePen.

选择器可被看做表达式，通过它可以选择相应的元素并应用不同的样式。

- 简单选择器
- 元素选择器
- 组合选择器

简单选择器

简单选择器可组合使用。

标签选择器

```
<div>
  <p>Sample Paragraph</p>
  <p>Sample Paragraph</p>
  <p>Sample Paragraph</p>
</div>

<style type="text/css">
  p {
    color: blue;
  }
</style>
```

类选择器

`.className` 以 `.` 开头，名称可包含字母，数字，`-`，`_`，但必须以字母开头。它区分大小写并可出现多次。

```
<div>
  <p>Sample Paragraph</p>
  <p class="special bold">Sample Paragraph</p>
  <p>Sample Paragraph</p>
</div>

<style type="text/css">
  p {
    color: blue
  }
  .special {
    color: orange;
  }
  .bold {
    font-weight: bold;
  }
</style>
```

id 选择器

`#idName` 以 `#` 开头且只可出现一次，其命名要求于 `.className` 相同。

```
<div>
  <p id="special">Sample Paragraph</p>
</div>

<style type="text/css">
  #special {
    color: red;
  }
</style>
```

通配符选择器

```
<div>
  <p>Sample Paragraph</p>
  <p>Sample Paragraph</p>
</div>

<style type="text/css">
  * {
    color: blue;
  }
</style>
```

属性选择器

`[attr]` 或 `[attr=val]` 来选择相应的元素。`#nav{...}` 既等同于 `[id=nav]{...}` 。IE7+

`[attr~=val]` 可选用与选择包含 `val` 属性值的元素，像 `class="title sports"` 与 `class="sports"` 。 `.sports{...}` 既等同于 `[class~=sports]{...}` IE7+

`[attr|=val]` 可以选择 `val` 开头及开头紧接 `-` 的属性值。IE7+

`[attr^=val]` 可选择以 `val` 开头的属性值对应的元素，如果值为符号或空格则需要使用引号 `""` 。IE7+

`[attr$=val]` 可选择以 `val` 结尾的属性值对应的元素。IE7+

`[attr*=val]` 可选择以包含 `val` 属性值对应的元素。IE7+

```
<div>
  <form action="">
    <input type="text" value="Xinyang" disabled>
    <input type="password" placeholder="Password">
    <input type="button" value="Button">
  </form>
</div>
<style type="text/css">
  [disabled] {
    background-color: orange;
  }
  [type=button] {
    color: blue;
  }
</style>
```

伪类选择器

常用伪类选择器：

- `:link` IE6+
- `:visited` IE7+
- `:hover` IE6中仅可用于链接
- `:active` IE6/7中仅可用于链接
- `:enabled` IE9+
- `:disabled` IE9+
- `:checked` IE9+
- `:first-child` IE8+
- `:last-child` IE9+
- `:nth-child(even)` 可为 `odd` `even` 或数字 IE9+
- `:nth-last-child(n)` `n` 从 0 开始计算 IE9+
- `:only-child` 仅选择唯一的元素 IE9+
- `:only-of-type` IE9+
- `:first-of-type` IE9+
- `:last-of-type` IE9+
- `:nth-of-type(even)` IE9+

- `:nth-last-of-type(2n)` IE9+

不常用伪类选择器：

- `:empty` 选中页面中无子元素的标签 IE9+
- `:root` 选择 HTML 根标签 IE9+
- `:not()` 参数为一般选择器 IE9+
- `:target` 被锚点选中的目标元素 IE9+
- `:lang()` 选中语言值为某类特殊值的元素 IE7+

NOTE： `element:nth-of-type(n)` 指父元素下第 n 个 `element` 元素， `element:nth-child(n)` 指父元素下第 n 个元素且元素为 `element` ，若不是，选择失败。具体细节请在使用时查找文档。

```
<div>
  <a href="http://sample-site.com" title="Sample Site">Sample Site</a>
</div>
<style type="text/css">
  /* 伪类属性定义有顺序要求！ */
  a:link {
    color: gray;
  }
  a:visited {
    color:red;
  }
  a:hover {
    color: green;
    /* 鼠标悬停 */
  }
  a:active {
    color: orange;
    /* 鼠标点击 */
  }
</style>
```

其他选择器

伪元素选择器

注意与伪类学则器的区分。

- `::first-letter` IE6+
- `::first-line` IE6+
- `::before{content: "before"}` 需与 `content` 一同使用 IE8+
- `::after{content: "after"}` 需与 `content` 一同使用 IE8+
- `::selection` 被用户选中的内容（鼠标选择高亮属性）IE9+ Firefox需用 `-moz` 前缀

组合选择器

- 后代选择器 `.main h2 {...}`，使用 ` ` 表示 IE6+
- 子选择器 `.main>h2 {...}`，使用 `>` 表示 IE7+
- 兄弟选择器 `h2+p {...}`，使用 `+` 表示 IE7+
    - `h2~p {...}`，使用 `~` 表示（此标签无需紧邻）IE7+

选择器分组

```
<style type="text/css">
/* 下面两组样式声明效果一致 */
h1 {color: red;}
h2 {color: red;}
h3 {color: red;}

h1, h2, h3 {color: red;}
</style>
```

继承、优先、层级

继承

子元素继承父元素的样式，但并不是所有属性都是默认继承的。通过文档中的 `inherited: yes` 来判断属性是否可以自动继承。

Initial value	Varies from one browser to another
Applies to	all elements. It also applies to ::first-letter and ::first-line.
Inherited	yes
Media	visual
Computed value	If the value is translucent, the computed value will be the rgba() corresponding one. If it isn't, it will be the rgb() corresponding one. The transparent keyword maps to rgba(0,0,0,0).
Animatable	yes, as a color
Canonical order	the unique non-ambiguous order defined by the formal grammar

自动继承属性：

- color
- font
- text-align
- list-style
- ...

非继承属性：

- background
- border
- position
- ...

优先

CSS Specificity Calculator 可以在这里找到。更多关于 CSS 优先级别的信息可以在这里找到（英文）。

计算方法：

- a = 行内样式
- b = id 选择器的数量
- c = 类、伪类的属性选择器的数量
- d = 标签选择器和伪元素选择器的数量

NOTE：从上到下优先级一次降低，且优先级高的样式会将优先级低的样式覆盖。大致公式（并不准确）如下。

```
value = a * 1000 + b * 100 + c * 10 + d
```

改变优先级

- 改变样式声明先后顺序
- 提升选择器优先级
- `!important` （慎用）

层叠

层叠为相同属性根据优先级覆盖，如优先级相同则后面会覆盖前面的属性，而不同属性则会合并。

Table of Contents *generated with DocToc*

文本

See the Pen FEND_Fonts by Li Xinyang (@li-xinyang) on CodePen.

字体

改变字号

```
font-size: <absolute-size> | <relative-size> | <length> | <percentage> | inherit
```

- `<absolute-size>` 有 small large medium

- `<relative-size>` 有 smaller larger

```
div
  font-size 12px
  p#sample0
    font-size 16px
  p#sample1
    font-size 2em
  p#sample2
    font-size 200%
```

NOTE：以上两值在开发中并不常用。 `2em` 与 `200%` 都为父元素默认大小的两倍（参照物为父元素的字体大小 `12px` ）。

改变字体

`font-family: [ <family-name> | <generic-family> ]#`

`<generic-family>` 可选选项，但具体使用字体由浏览器决定

- serif
- sans-serif
- cursive
- fantasy
- monospace

```
font-family: arial, Verdana, sans-serif;
```

NOTE：优先使用靠前的字体

加粗字体

`font-weight: normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900`

```
font-weight: normal;
font-weight: bold;
```

倾斜字体

```
font-style: normal | italic | oblique | inherit
```

`italic` 使用字体中的斜体，而 `oblique` 在没有斜体字体时强制倾斜字体。

更改行距

```
line-height: normal | <number> | <length> | <percentage>
```

`normal` 值为浏览器决定，在1.1至1.2之间（通常设置值为1.14左右）

```
/* length 类型 */
line-height: 40px;
line-height: 3em;
/* percentage 类型 */
line-height: 300%;
/* number 类型 */
line-height: 3;
```

NOTE：当 `line-height` 为 `number` 类型时，子类直接继承其数值（不计算直接继承）。而当为 `percentage` 类型时，子类则会先计算再显示（先计算后继承）。

字间距（字母间距）

```
letter-spacing: normal | <length>
```

其用于设置字间距或者字母间距，此属性适用于中文或西文中的字母。如果需要设置西文中词与词的间距或标签直接的距离则需要使用 `word-spacing` 。

```
word-spacing: normal | <length>
```

font shorthand

```
font: [ [ <‘font-style’> || <font-variant-css21> || <‘font-weight’> || <‘font-stretch’> ]? <‘font-size’> [ / <‘line-height’> ]? <‘font-family’> ] | caption | icon | menu | message-box | small-caption | status-bar
```

```
font: 30px/2 "Consolas", monospace;
font: italic bold 20px/1.5 arial, serif;
font: 20px arial, serif;
```

NOTE：当其他值为空时，均被设置为默认值。

改变文字颜色

`color: <color>`

```
element { color: red; }
element { color: #f00; }
element { color: #ff0000; }
element { color: rgb(255,0,0); }
element { color: rgb(100%, 0%, 0%); }
element { color: hsl(0, 100%, 50%); }

/* 50% translucent */
element { color: rgba(255, 0, 0, 0.5); }
element { color: hsla(0, 100%, 50%, 0.5); }

/* 全透明 */
element { color: transparent' }
element { color: rgba(0, 0, 0, 0); }
```

对齐方式

文字居中

`text-align: start | end | left | right | center | justify | match-parent | start end`

NOTE：默认为文本左对齐。

文本垂直对齐

`vertical-align: baseline | sub | super | text-top | text-bottom | middle | top | bottom | <percentage> | <length>`

NOTE： `<percentage>` 的参照物为 `line-height`

文本缩进

`text-indent: <length> | <percentage> && [ hanging || each-line ]`

NOTE：缩进两个字可使用 `text-indent: 2em;`

格式处理

保留空格格式

`white-space: normal | pre | nowrap | pre-wrap | pre-line`

`pre` 行为同 `<pre>` 一致。

	New lines	Spaces and tabs	Text wrapping
`normal`	Collapse	Collapse	Wrap
`nowrap`	Collapse	Collapse	No wrap
`pre`	Preserve	Preserve	No wrap
`pre-wrap`	Preserve	Preserve	Wrap
`pre-line`	Preserve	Collapse	Wrap

文字换行

`word-wrap: normal | break-word`

NOTE：允许长单词自动换行。

`word-break: normal | break-all | keep-all`

NOTE：`break-all` 单词中的任意字母间都可以换行。

文本装饰

文字阴影

`text-shadow:none | <shadow-t>#` 或 `text-shadow:none | [<length>{2,3}&&<color>?]#`

```
p {
  text-shadow: 1px 1px 1px #000,
               3px 3px 5px blue;
}
```

1. value = The X-coordinate X 轴偏移像素
2. value = The Y-coordinate Y 轴偏移像素
3. value = The blur radius 阴影模糊半径

4. value = The color of the shadow 阴影颜色（默认为文字颜色）

文本装饰（下划线等）

`text-decoration: <'text-decoration-line'> || <'text-decoration-style'> || <'text-decoration-color'>`

```
h1.under {
    text-decoration: underline;
}
h1.over {
    text-decoration: overline;
}
p.line {
    text-decoration: line-through;
}
p.blink {
    text-decoration: blink;
}
a.none {
    text-decoration: none;
}
p.underover {
    text-decoration: underline overline;
}
```

高级设置

省略字符

`text-overflow: [ clip | ellipsis | <string> ]{1,2}`

```
/* 常用配合 */
text-overflow: ellipsis;
overflow: hidden; /* 溢出截取 */
white-space: nowrap; /* 禁止换行 */
```

更换鼠标形状

`cursor: [[<funciri>,]* [ auto | crosshair | default | pointer | move | e-resize | ne-resize | nw-resize | n-resize | se-resize | sw-resize | s-resize | w-resize| text | wait | help ]] | inherit`

常用属性

`cursor: [<uri>,]*[auto | default | none | help | pointer | zoom-in | zoom-out | move]`

- `<uri>` 图片资源地址代替鼠标默认形状
- `<default>` 默认光标
- `<none>` 隐藏光标
- `<pointer>` 手型光标
- `<zoom-in>`
- `<zoom-out>`
- `<move>`

```
cursor: pointer;
cursor: url(image-name.cur), pointer;
/* 当 uri 失效时或者则会起作用 */
```

强制继承

`inherit` 会强制继承父元素的属性值。

```
font-size: inherit;
font-family: inherit;
font-weight: inherit;
...
word-wrap: inherit;
work-break: inherit
text-showdow: inherit
```

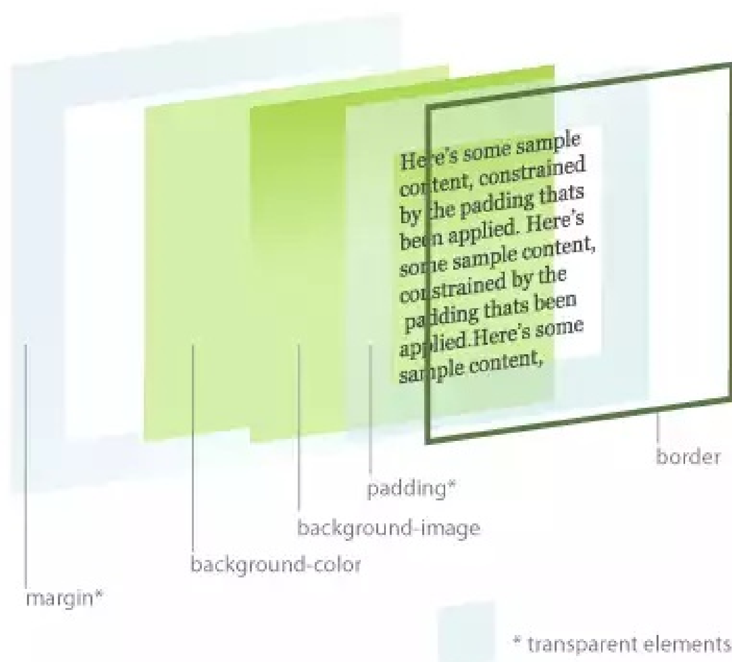
NOTE：具体在使用时可查询文档

Table of Contents *generated with DocToc*

盒模型

盒子模型是网页布局的基石。它有边框、外边距、内边距、内容组成。

盒子 **3D** 模型

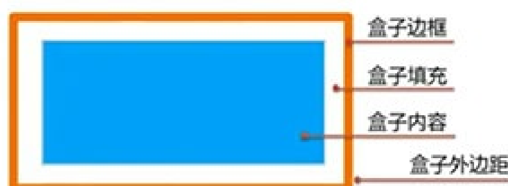


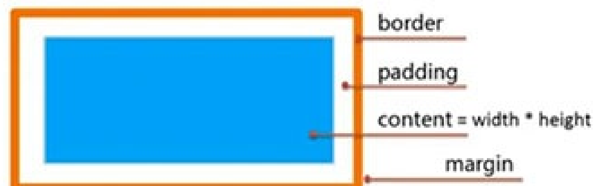
盒子由上到下依次分为五层，它们自上而下的顺序是。

1. border 边框
2. content + padding 内容与内边距
3. background-image 背景图片
4. background-color 背景颜色
5. margin 外边距

See the Pen FEND_003_BoxModel by Li Xinyang (@li-xinyang) on CodePen.

属性





width

内容盒子宽

`width: <length> | <percentage> | auto | inherit`

NOTE：通常情况下百分比得参照物为元素的父元素。 `max-width` 与 `min-width` 可以设置最大与最小宽度。

height

内容盒子高

`height: <length> | <percentage> | auto | inherit`

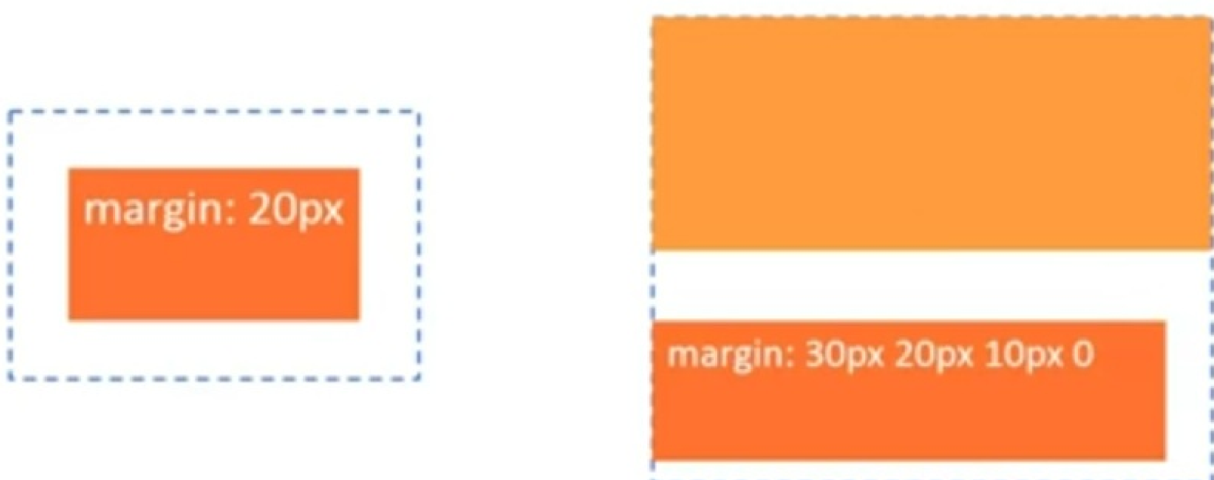
NOTE：默认情况元素的高度为内容高度。 `max-height` 与 `min-height` 可以设置最大与最小高度。

padding



`padding: [<length> | <percentage>]{1,4} | inherit`

margin



`margin: [<length> | <percentage> | auto]{1,4} | inherit`

NOTE： `margin` 默认值为 `auto`

Trick：

```
/* 可用于水平居中 */
margin: 0 auto;
```

margin 合并

margin-bottom: 40px;

margin-top: 20px;

毗邻元素

margin-bottom: 40px;

margin-bottom: 20px;

父元素与第一个/最后一个子元素

毗邻元素外间距（margin）会合并，既取相对较大的值。父元素与第一个和最后一个子元素的外间距也可合并。

border

border: 1px dashed blue;

border-width: 0 1px 2px 3px;
border-style: solid;
border-color: #0ff;

```
border: [<br-width> || <br-style> || <color>] | inherit
border-width: [<length> | thin | medium | thick]{1,4} | inherit
border-style: [solid | dashed | dotted | ...]{1,4} |inherit
border-colro: [<color> | transparent]{1,4} | inherit
```

NOTE： `border-color` 默认为元素字体颜色。

border-radius

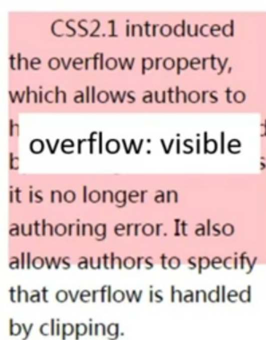


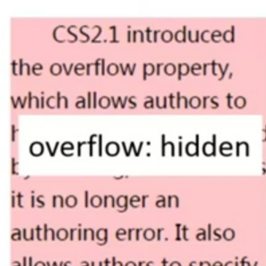


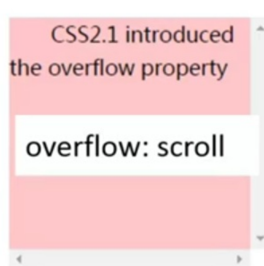

```
/* 水平半径/垂直半径 */
border-radius: [ <length> | <percentage> ]{1,4} [ / [ <length> |
 <percentage> ]{1,4} ]?
```

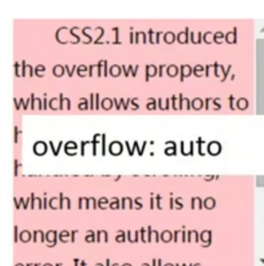
NOTE：四个角的分解属性由左上角顺时针附值。

overflow





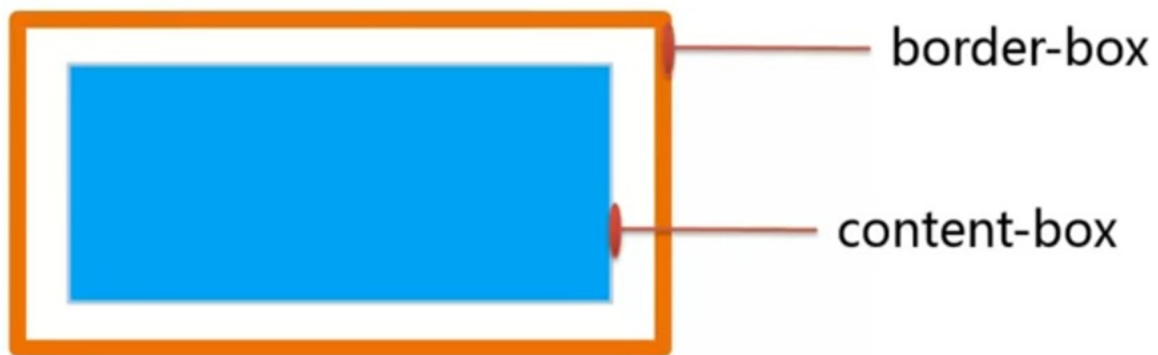




`overflow: visible | hidden | scroll | auto`

NOTE：默认属性为 `visible` 。使用 `overflow-x` 与 `overflow-y` 单独的设置水平和垂直方向的滚动条。

box-sizing








```
box-sizing: content-box | border-box | inherit
```

- `content-box` = 内容盒子宽高 + 填充（ `Padding` ）+ 边框宽（ `border-width` ）
- `border-box` = 内容盒子宽高

box-shadow



外阴影



内阴影



多阴影

`box-shadown: none | [inset? && [ <offset-x> <offset-y> <blur-radius>? <spread-radius>? <color>? ] ]#`

```
box-shadow: 4px  6px   3px  0px red;
             |    |     |    |
        水平偏移|     |    |
             垂直偏移 |    |
                  模糊半径 |
                       阴影大小
```

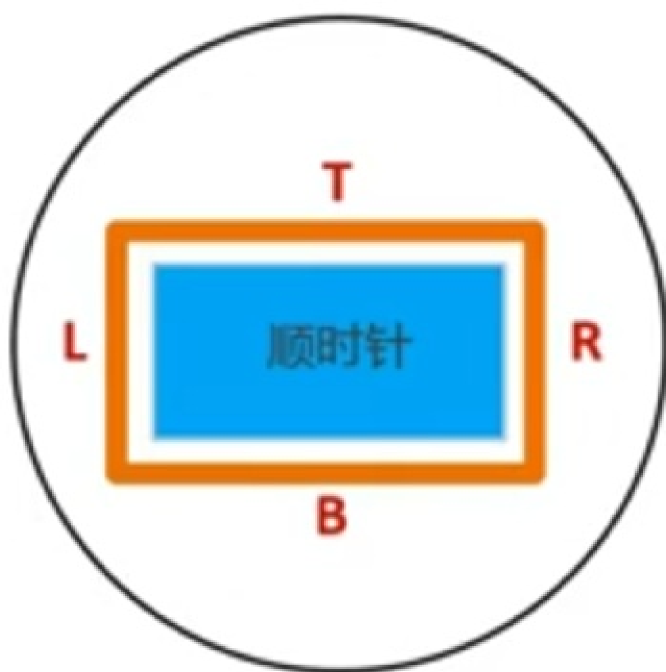
NOTE：水平与垂直偏移可以为负值即相反方向偏移。颜色默认为文字颜色。阴影不占据空间，仅为修饰效果。

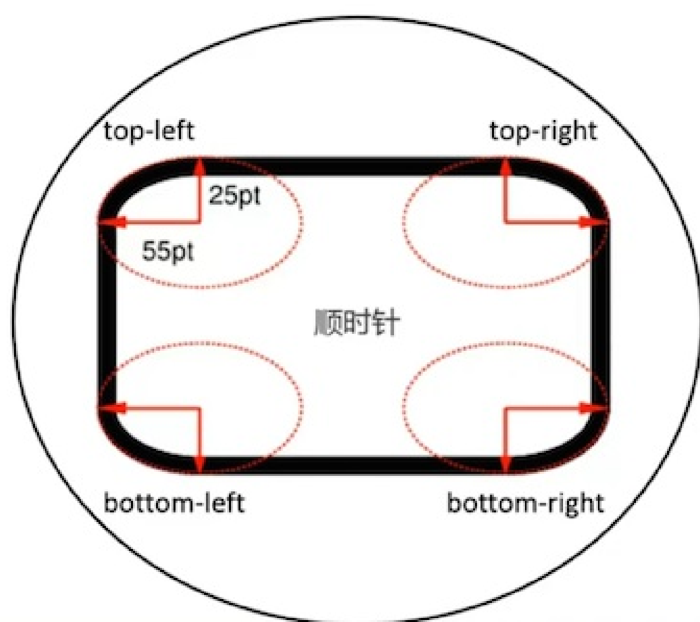
outline

```
outline: [ <'outline-color'> || <'outline-style'> || <'outline-width'> ]
outline-width: <length> | thin | medium | thick | inherit
outline-style: solid | dashed | dotted | ... | inherit
outline-color: <color> | invert | inherit
/* invert 与当前颜色取反色 */
```

NOTE： `outline` 与 `border` 相似但无法分别设置四个方向的属性。 `outline` 并不占据空间，而 `border` 占据空间，且显示位于 `border` 以外。

TRBL





TRBL (Top, Right, Bottom, Left) 即为顺时针从顶部开始。具有四个方向的属性都可以通过 `*-top` `*-right` `*-bottom` 与 `*-left` 单独对其进行设置。

值缩写

下面的值缩写以 `padding` 为例。

> 对面相等，后者省略；四面相等，只设一个。

```
/*      四面值 */
padding: 20px;
padding: 20px 20px 20px 20px;

/*      上下值 右左值 */
padding: 20px   10px;
padding: 20px 10px 20px 10px;

/*       上值 右左值 下值 */
padding: 20px 10px   30px;
padding: 20px 10px 30px 10px;
```

Table of Contents *generated with [DocToc]*

背景

background-color

```
background-color: <color>
background-color: #f00;
background-color: rgba(255, 0, 0, 0.5);
background-color: transparent; /* 默认值 */
```

background-image

```
background-image: <bg-image>[, <bg-image>]*
/* <bg-image> = <image> | none */
background-image: url("../image/pic.png");
background-image: url("../image/pic.png0"), url("../image/pic1.p
ng");
/* 多张背景图时，先引入的图片在上一层后引入则在下一层 */
```

NOTE：当 `background-color` 与 `background-image` 共存时，背景颜色永远在最底层（于背景图片之下）。

background-repeat

`background-repeat` 需与背景图片数量一致。

```
background-repeat: <repeat-style>[, <repeat-style]*
<repeat-style> = repeat-x | repeat-y | [repeat | space | round |
 no-repeat]{1,2}

/*                   X 轴     Y 轴 */
background-repeat: no-repeat repeat;
```

- `space` 平铺并在水平和垂直留有空隙，空隙的大小为图片均匀分布后完整覆盖显示区域的宽高
- `round` 不留空隙平铺且覆盖显示区域，图标会被缩放以达到覆盖效果（缩放不一定等比）

background-attachment

当页面内容超过显示区域时，使用 `local` 使背景图片同页面内容一同滚动。

```
background-attachment: <attachment>[, <attachment>]*
<attachment> = scroll | fixed | local
```

background-position

```
background-position: <position>[, <position>]*
<position> = [left|center|right|top|bottom|<percentage>|<length>
]|[left|center|right|top|bottom|<percentage>|<length>] [left|cen
ter|right|top|bottom|<percentage>|<length>] | [center |[left|rig
ht][<percentage>|<length>]?]&&[center |[left|right][<percentage>
|<length>]?]

/* 默认位置为 */
background-position: 0 0;

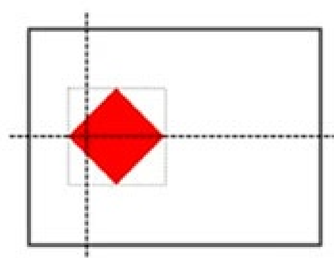
/* percentage 是容器与图片的百分比重合之处*/
background-position: 20% 50%;

/* 等同效果 */
background-position: 50% 50%;
background-position: center center;

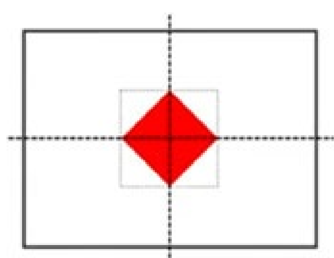
background-position: 0 0;
background-position: left top;

background-position: 100% 100%;
background-position: right bottom;

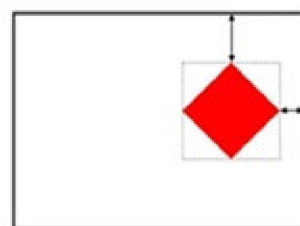
/* 四个值时方向只为参照物 */
background-position: right 10px top 20px;
```



20% 50%



50% 50%



right 10px top 20px

Sprite 的使用

```
background-image: url(sprite.png)
background-repeat: no-repeat;
background-positon: 0 -100px
```

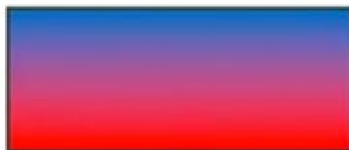
使用位置为负值将图片偏移使需要的图片位置上移并显示正确的图案。

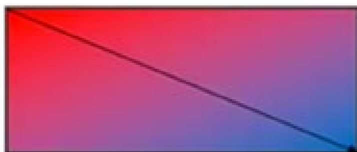
linear-gradient

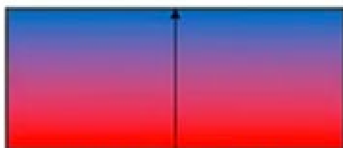
```
linear-gradient()
[[<angle> | to <side-or-corner],]? <color-step>[, <color-stop>]+
<side-or-corner> = [left | right] || [top | bottom]
<color-stop> = <color> [<percentage> | <length>]?

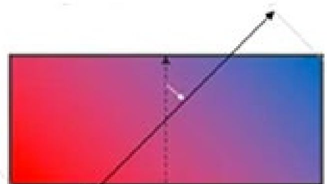
background-image: linear-gradient(red, blue);
background-image: linear-gradient(to top, red, blue);
background-image: linear-gradient(to right bottom, red, blue);
background-image: linear-gradient(0deg, red, blue);
background-image: linear-gradient(45deg, red, blue);
background-image: linear-gradient(red, green, blue);
background-image: linear-gradient(red, green 20%, blue);
```

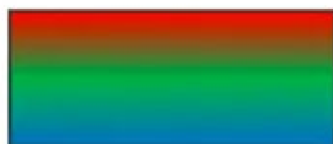

(red, blue)


(to top, red, blue)


(to right bottom, red, blue)


(0deg, red, blue)


(45deg, red, blue)


(red, green, blue)

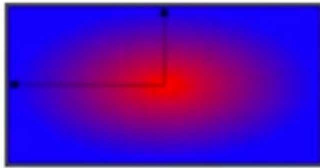

(red, green 20%, blue)

radial-gradient

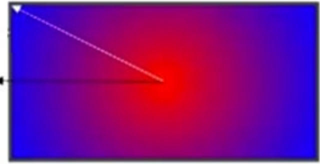
```
radial-gradient(   [ circle || <length> ] [ at <position> ]? , |
 [ ellipse || [<length> | <percentage> ]{2}] [ at <position> ]?
, | [ [ circle | ellipse ] || <extent-keyword> ] [ at <position>
 ]? , | at <position> , <color-stop> [ , <color-stop> ]+ )

<extent-keyword> = closest-corner | closest-side | farthest-corn
er | farthest-side
<color-stop> = <color> [ <percentage> | <length> ]?

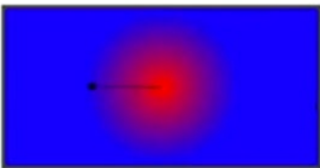
background-image: radial-gradient(cloest-side, red, blue);
background-image: radial-gradient(circle, red, blue);
background-image: radial-gradient(circle 100px, red, blue);
background-image: radial-gradient(red, blue);
background-image: radial-gradient(100px 50px, red, blue);
background-image: radial-gradient(100px 50px at 0 0, red, blue);
background-image: radial-gradient(red, green 20%, blue);
```



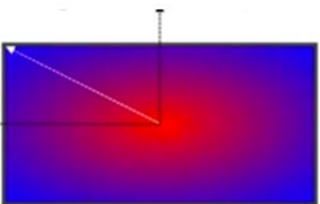
(cloest-side, red, blue)



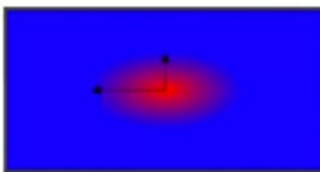
(circle, red, blue)



(circle 100px, red, blue)



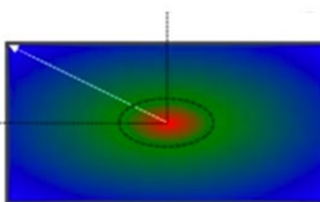
(red, blue)



(100px 50px, red, blue)



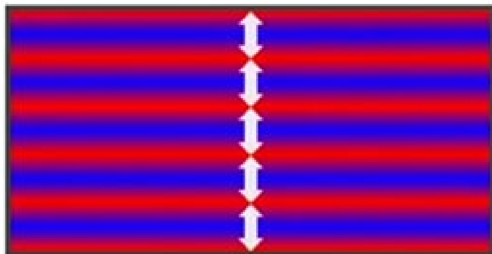
(100px 50px at 0 0, red, blue)



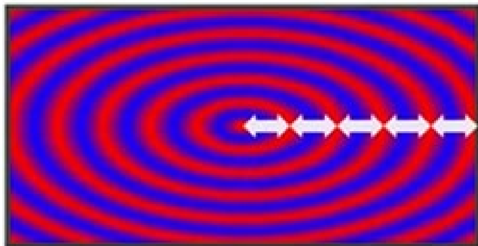
(red, green 20%, blue)

repeat-*-gradient

```
background-image: repeating-linear-gradient(red, blue 20px, red
40px);
background-image: repeating-radial-gradient(red, blue 20px, red
40px);
```



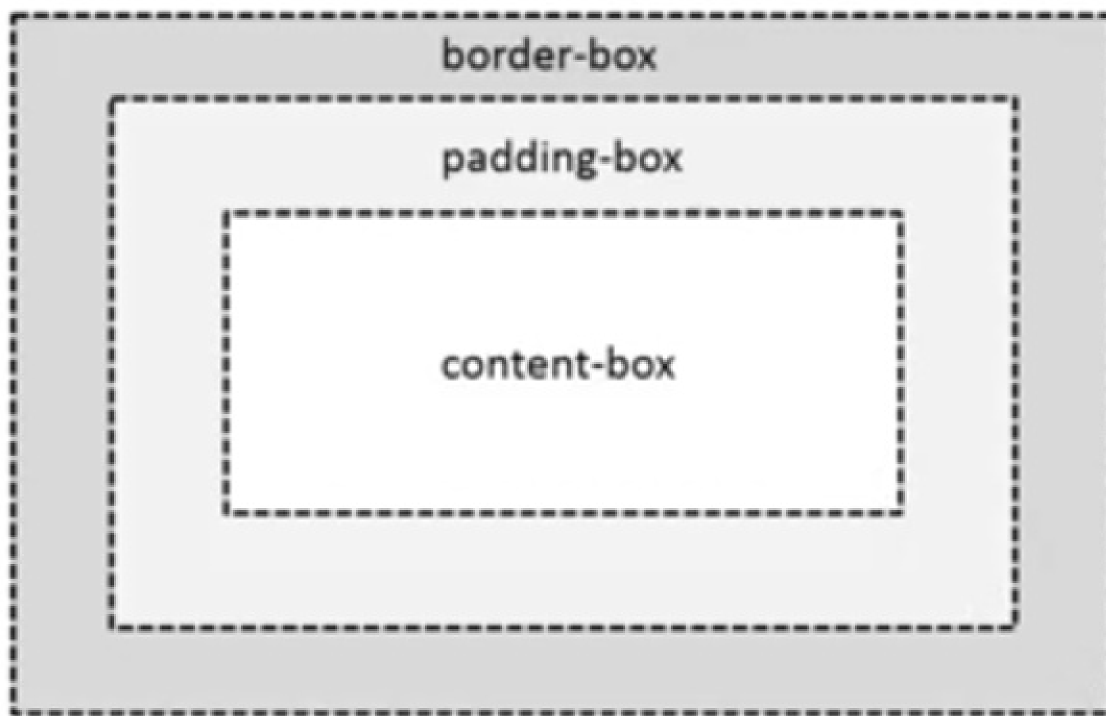
repeating-linear-gradient



repeating-radial-gradient

background-origin

案例模型

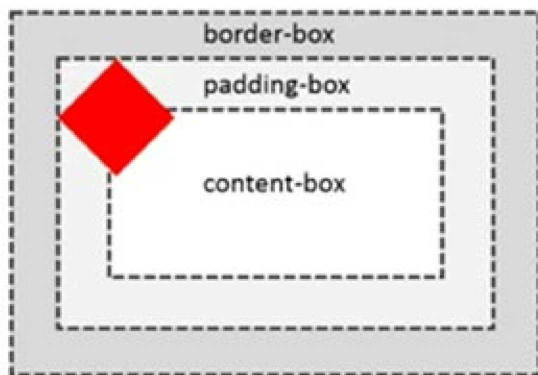


决定背景 (0,0) 坐标与 100% 坐标的区域。默认值为 `padding-box` 。

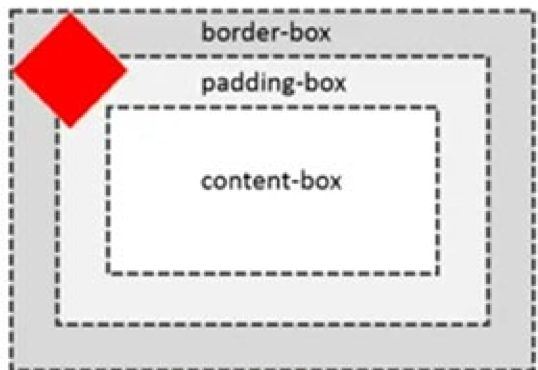
```
<box>[, <box>]*
<box> = border-box | padding-box | content-box

background-image: url(red.png);
background-repeat: no-repeat;

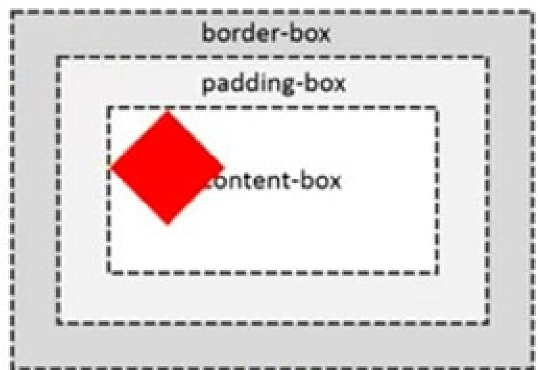
background-origin: padding-box;
background-origin: border-box;
background-origin: content-box;
```



padding-box



border-box



content-box

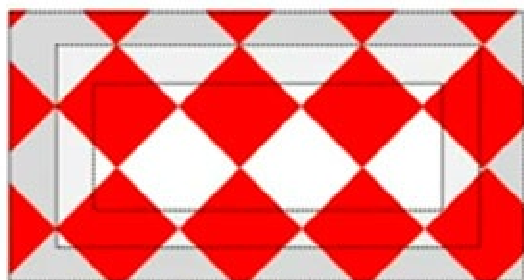
background-clip

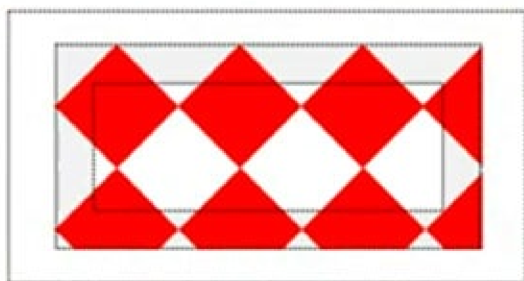
裁剪背景，默认值为 `border-box` 。

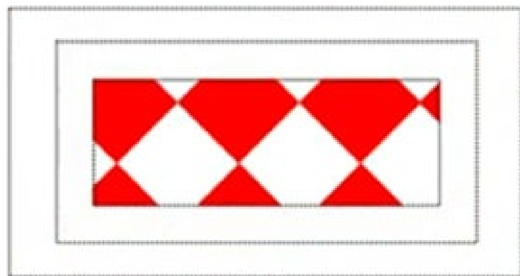
```
<box>[, <box>]*
<box> = border-box | padding-box | content-box

background-image: url(red.png);
background-repeat: no-repeat;

background-clip: border-box;
background-clip: padding-box;
background-clip: content-box;
```







background-size

```
<bg-size>[, <bg-size>]*
<bg-size> = [<length> | <percentage> | auto] {1, 2} | cover | co
ntain

background-image: url(red.png);
background-repeat: no-repeat;
background-position: 50% 50%;

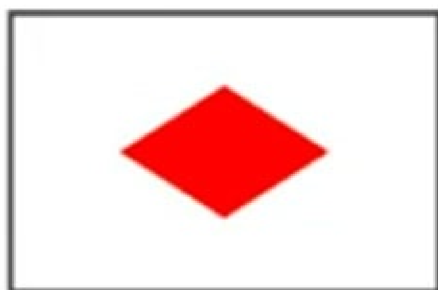
background-size: auto;
background-size: 20px 20px;
/* % 参照物为容器*/
background-size: 50% 50%;
/* 尽可能小，但宽度与高度不小过容器（充满容器） */
background-size: cover;
/* 尽可能大，但宽度与高度不超过容器（最大完全显示）*/
background-size: contain;
```



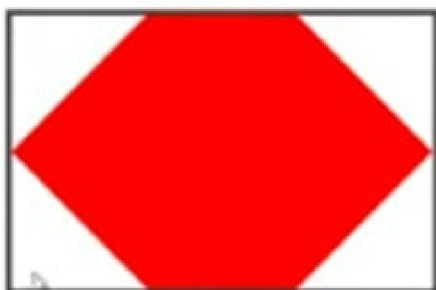
auto



20px 20px



50% 50%



cover



contain

background shorthand

```
[<bg-layer,]* <final-bg-layer>
<bg-layer> = <bg-image> || <position> [/ <bg-size>]? || <repeat-
style> || <attachment> || <box> || <box>

/* 两个 <box> 第一个为 background-origin */
/* 两个 <box> 第二个为 background-clip */
/* 只出现一个 <box> 则即是 background-origin 也是 background-clip *
/

<final-bg-layer> = <bg-layer> || <'background-color'>

background: url(red.png) 0 0/20px 20px no-repeat, url(blue.png)
50% 50%/contain no-repeat content-box green;
```

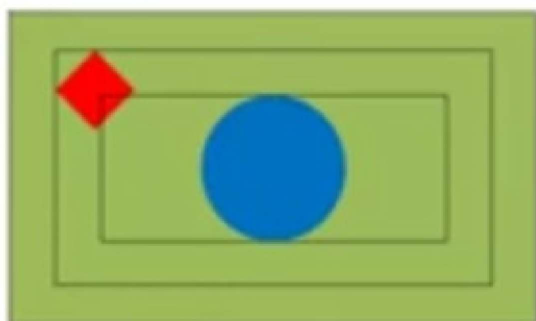


Table of Contents *generated with [DocToc]*

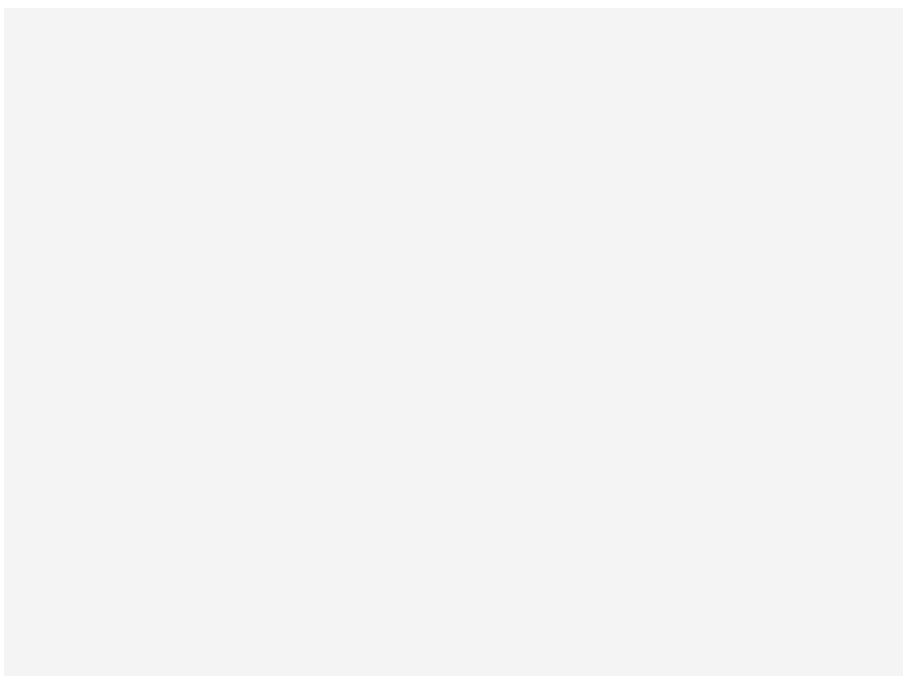
布局

学习布局前须知道 CSS 中的定位机制。

- 标准文档流（Normal Flow）
- 浮动（Float）
- 绝对定位（Absolute Positioning)

标准文档流，从上到下，从左到右的输出文档内容。它由块级（block）元素和行级元素组成，且它们都是盒子模型。

下面为 Firefox 布局可视化 **Gecko Reflow Visualisation**，布局是指浏览器将元素以正确的大小摆放在正确的位置上。



display

设置元素的显示方式

display	默认宽度	可设置宽高	起始位置
block	父元素宽度	是	换行
inline	内容宽度	否	同行
inline-block	内容宽度	是	同行

display:block

- 默认宽高为父元素宽高
- 可设置宽高

- 换行显示
- 默认为block的元素：`<div>`，`<p>`，`<h1>` ~ `<h6>`，`<ul>`，`<form>`

display:inline

- 默认宽度为内容宽度
- 不可设置宽高
- 同行显示（元素内部可换行）
- 默认为inline的元素：`<span>`，`<a>`，`<label>`，`<cite>`，`<em>`

display:inline-block

- 默认宽度为内容宽度
- 可设置宽高
- 同行显示
- 整块换行
- 默认为inline-block的元素：`<input>`，`<textarea>`，`<select>`，`<button>`

display:none

- 设置元素不显示

`display:none` 与 `visibility:hidden` 的区别为 `display:none` 不显示且不占位，但 `visibility:hidden` 不显示但占位。

position

`position` 用于设置定位的方式与 `top` `right` `bottom` `left` `z-index` 则用于设置参照物位置（必须配合定位一同使用）。

三种定位形式

- 静态定位（static）
- 相对定位（relative）
- 绝对定位（absolute、fixed）

```
position: static | relative | absolute | fixed
/* 默认值为 static */
```

position:relative

- 相对定位的元素仍在文档流之中，并按照文档流中的顺序进行排列。
- 参照物为元素本身的位置。

NOTE：最常用的目的为改变元素层级和设置为绝对定位的参照物。



position:absolute

建立以包含块为基准的定位，其随即拥有偏移属性和 `z-index` 属性。

- 默认宽度为内容宽度
- 脱离文档流
- 参照物为第一个定位祖先或根元素（ `<html>` 元素）



position:fixed

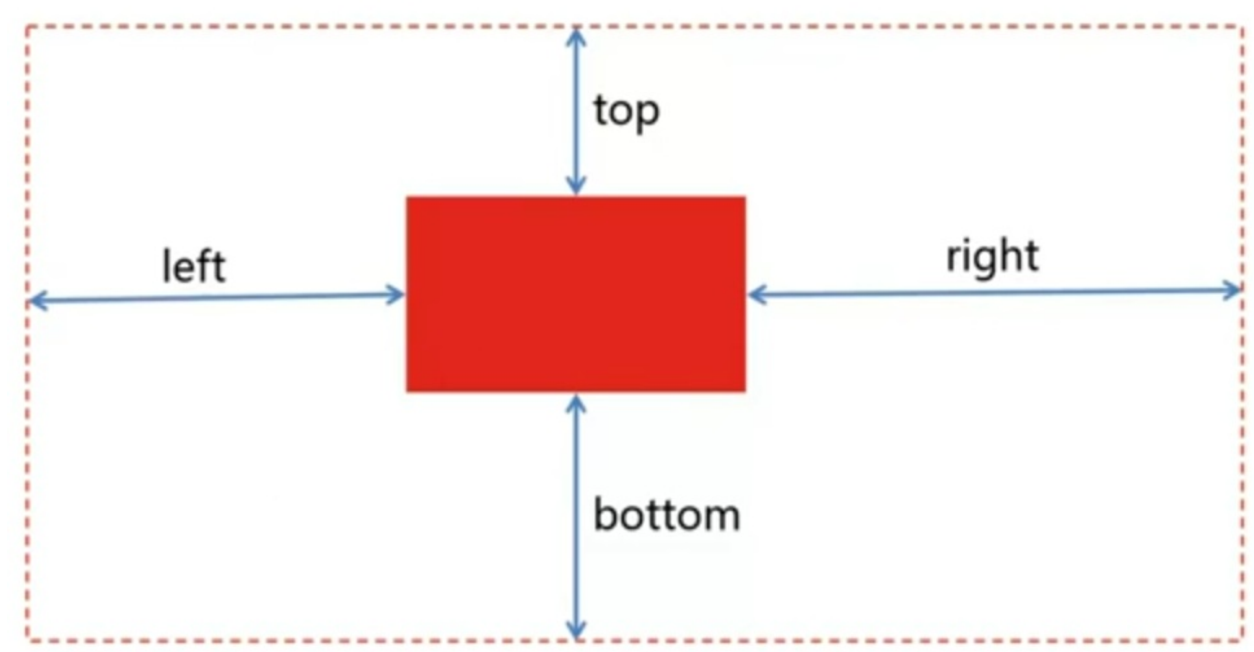
- 默认宽度为内容宽度
- 脱离文档流
- 参照物为视窗

NOTE：宽高的100%的参照依然为视窗（例：网页遮罩效果）





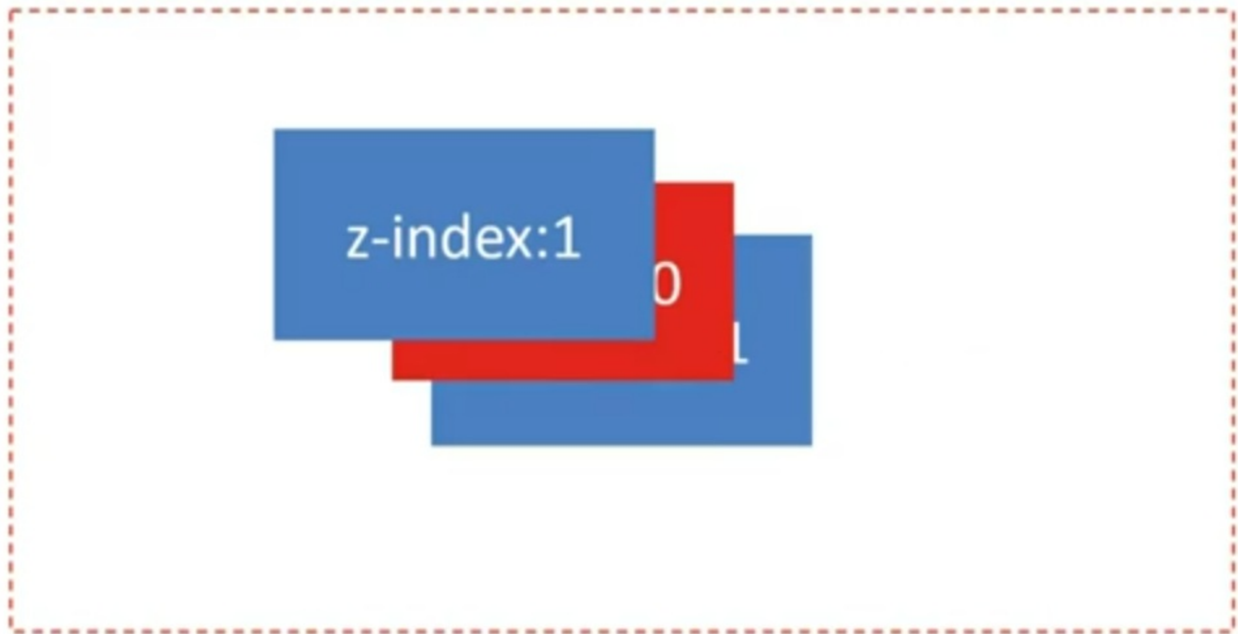
top/right/bottom/left



其用于设置元素边缘与参照物边缘的距离，且设置的值可为负值。在同时设置相对方向时，元素将被拉伸。

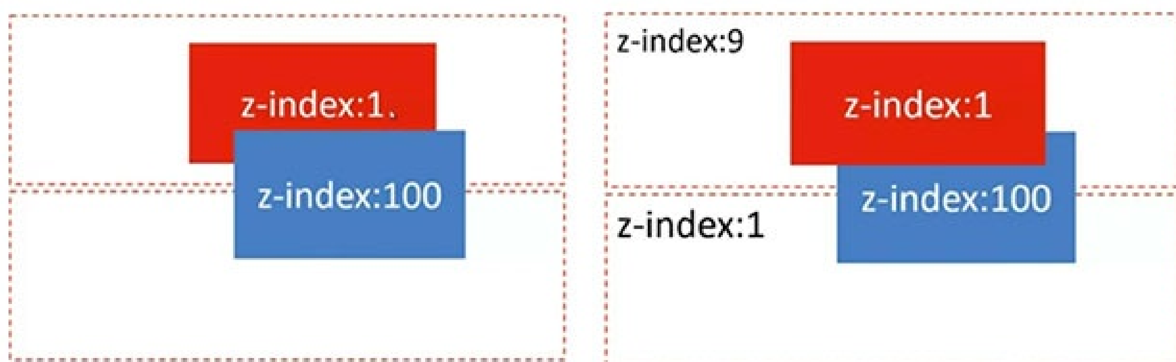
z-index

其用于设置 Z 轴上得排序，默认值为 0 但可设置为负值。（如不做设置，则按照文档流的顺序排列。后面的元素将置于前面的元素之上）



z-index 栈

父类容器的 `z-index` 优于子类 `z-index` 如图

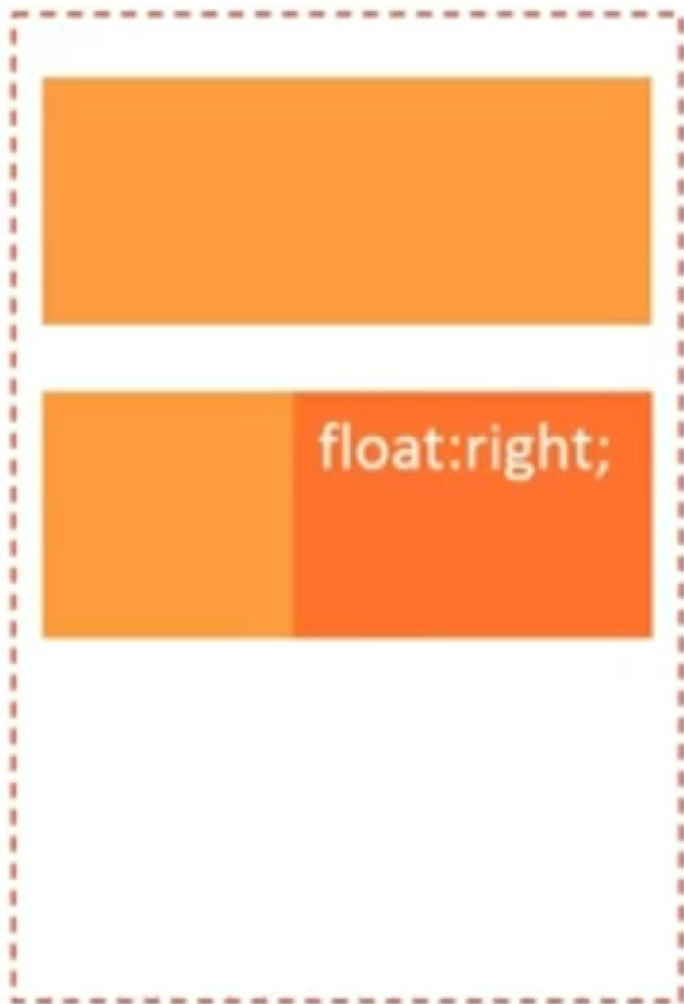


float

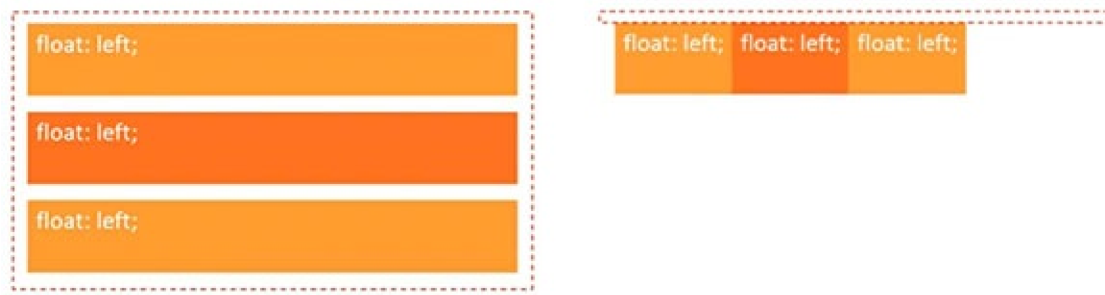
CSS 中规定的定位机制，其可实现块级元素同行显示并存在于文档流之中。浮动仅仅影响文档流中下一个紧邻的元素。

```
float: left | right | none | inherit
```

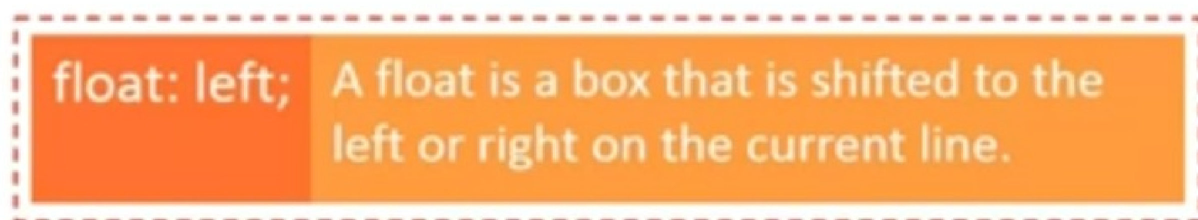
- 默认宽度为内容宽度
- 脱离文档流（会被父元素边界阻挡与 `position` 脱离文档流的方式不同）
- 指的方向一直移动



float 元素在同一文档流中，当同时进行 `float` 时它们会按照文档流中的顺序排列。(当所有父元素中的所有元素脱离文档流之后，父元素将失去原有默认的内容高度)



注意：**float** 元素是半脱离文档流的，对元素是脱离文档流，但对于内容则是在文档流之中的（既元素重叠但内容不重叠）。



clear

```
clear: both | left | right | none | inherit
```

- 应用于后续元素
- 应用于块级元素（block）

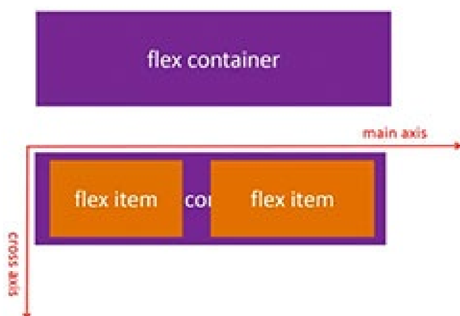
使用方法：

优先级自上而下

1. clearfix 于父元素
2. 浮动后续空白元素 `.emptyDiv {clear: both}`
3. 为受到影响的元素设置 `width: 100% overflow: hidden` 也可
4. 块级元素可以使用 `<br>` 不建议使用，影响 HTML 结构

```
/* clearfix */
.clearfix:after {
   content: "."; /* Older browser do not support empty content *
/
   visibility: hidden;
   display: block;
   height: 0;
   clear: both;
}
.clearfix {zoom: 1;} /* 针对 IE 不支持 :after */
```

flex



弹性布局可用于多行自适应，多列自适应，间距自适应和任意对齐。

创建 **flex container**

```
display: flex
/* 弹性容器内的均为弹性元素*/
```

flex item

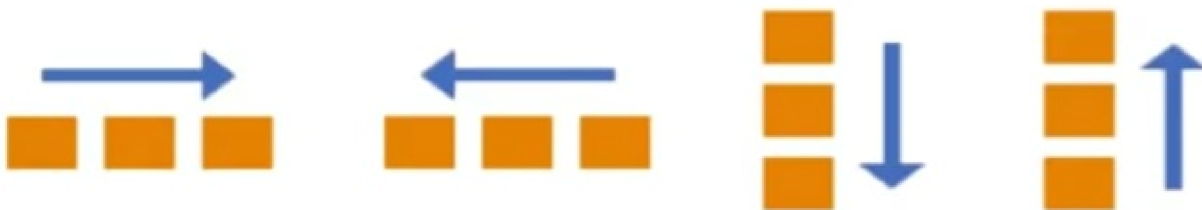
只有弹性容器在文档流中的子元素才属于弹性元素。

```
<div style="display: flex;">
  <div>Block Element</div>
  <!-- flex item: YES-->
  <span>Inline Element</span>
  <!-- flex item: YES-->
  <div style="position:absolute;">Absolute Block Element</div>
  <!-- flex item: YES-->
</div>
```

flex 方向

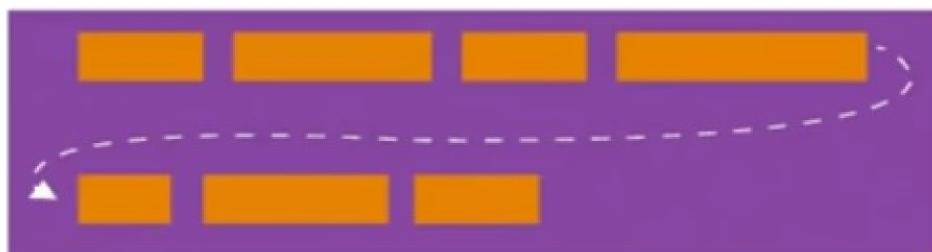
flex-direction

```
<!-- 默认值为 row -->
flex-direction: row | row-reverse | column | column-reverse
```



flex-wrap

```
<!-- 默认值为 nowrap -->
flex-wrap: nowrap | wrap | wrap-reverse
```



flex-flow

`flex-flow` 为 `flex-wrap` 与 `flex-direction` 的简写，建议使用此属性（避免同时使用两个属性来修改）。

```
flex-flow: <'flex-direction'> || <'flex-wrap'>
```



order

`order` 的值为相对的（同被设置和未被设置的值相比较），当均为设置时默认值为 0 则按照文档流中的顺序排列。

```
order: <integer>
<!-- 默认为 0 -->
```



flex 弹性

flex-basis

共用于设置 `flex-item` 的初始宽高（并作为弹性的基础）。如果 `flex-direction` 是以 `row` 排列则设置宽，如以 `column` 排列则设置高。

```
flex-basis: main-size | <width>
```

flex-grow

伸展因子，其为弹性布局中最重要的元素之一， `flex-grow` 设置元素可用空余空间的比例。 `flex-container` 先安装宽度（ `flex-basis` ）进行布局，如果有空余空间就按照 `flex-grow` 中的比例进行分配。

Width/Height = flex-basis + flex-grow/sum(flow-grow) * remain

```
flex-grow: <number>
initial: 0
<!-- 默认值为 0 -->
```



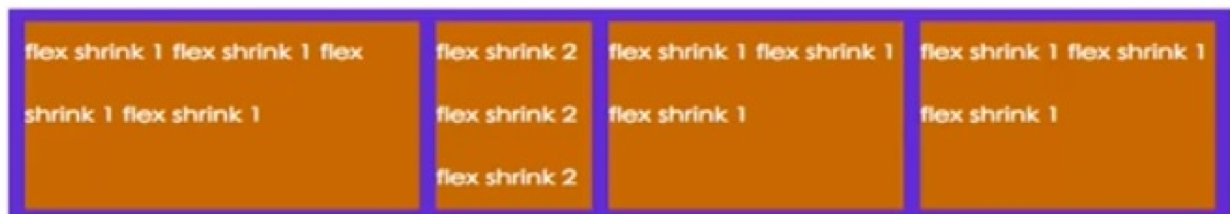
flex-shrink

收缩因子，用于分配超出的负空间如何从可用空间中进行缩减。

```
flex-shrink: <number>
initial: 1
<!-- 默认值为 1 -->
```

Width/Height = flex-basis + flow-shrink/sum(flow-shrink) * remain

remain 为负值，既超出的区域。



flex

其为 `flex-grow` `flex-shrink` `flex-basis` 的值缩写。

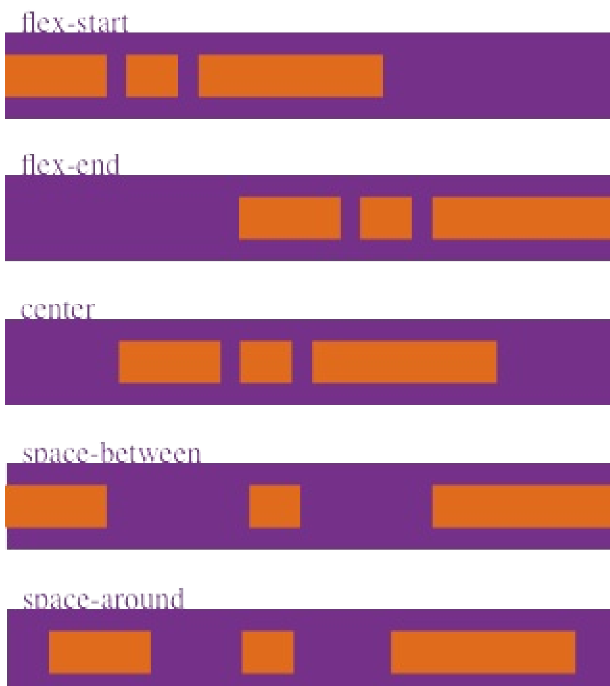
```
flex: <'flex-grow'> || <'flex-shrink'> || <'flex-basis'>
initial: 0 1 main-size
```

flex 对齐

justify-content

其用于设置主轴（main-axis）上的对其方式。弹性元素根据主轴（横向和纵向均可）定位所以不可使用 `left` 与 `right` 因为位置为相对的。（行为相似的属性有 `text-align` ）

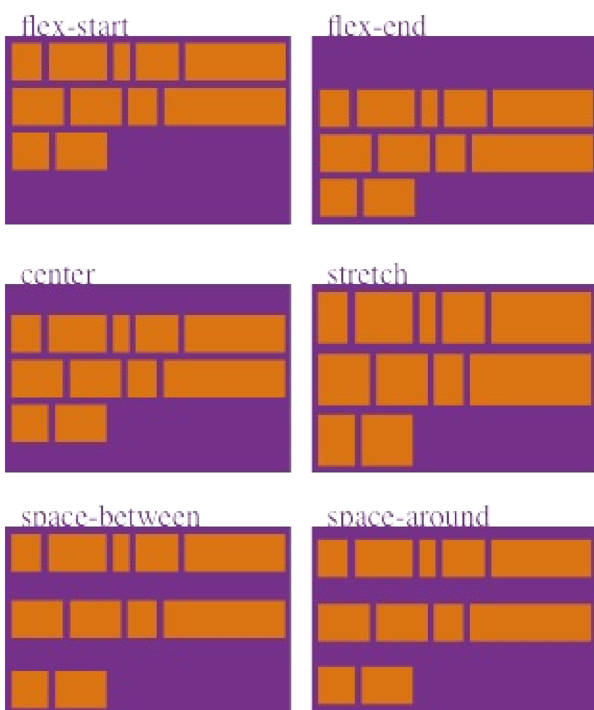
```
justify-content: flex-start | flex-end | center | space-between
| space-around
<!-- 默认值为 flex-start -->
```



align-items

其用于设置副轴（cross-axis）上的对其方式。（行为相似的属性有 `vertical-align` ）

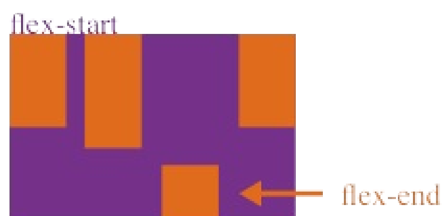
```
align-items: flex-start | flex-end | center | baseline | stretch
<!-- 默认值为 stretch -->
```



align-self

其用于设置单个 `flex-item` 在 cross-axis 方向上的对其方式。

```
align-self: auto | flex-start | flex-end | center | baseline | s
tretch
<!-- 默认值为 auto -->
```



align-content

其用于设置 cross-axis 方向上的对其方式。

```
align-content:flex-start | flex-end | center | space-between | s
pace-around | stretch
<!-- 默认为 stretch -->
```

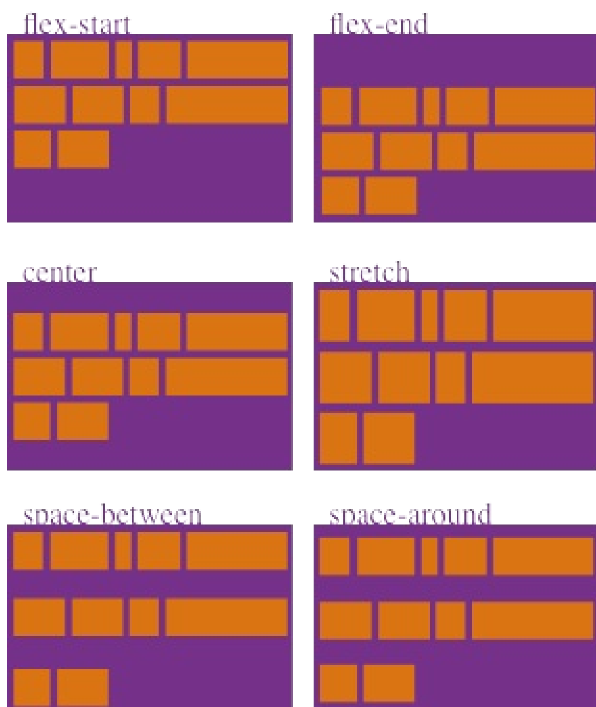


Table of Contents *generated with DocToc*

变形

2D 变形

2D 变形示例代码

transform

`transform` 中可以写一个或多个方法。

```
transform: none | <transform-function>+
transform: none
<!-- 默认值为 none -->
transform: <transform-function>+

transform: translate(50%) rotate(45deg);
transform: rotate(45deg) transform(50%)
<!-- 变形函数顺序普通结果不同，原因是坐标位置发生了改变 -->
```





rotate()

```
rotate(<angle>)

rotate(45deg);
<!-- 右边旋转，顺时针 -->
rotate(-60deg);
<!-- 左边旋转，逆时针 -->
```



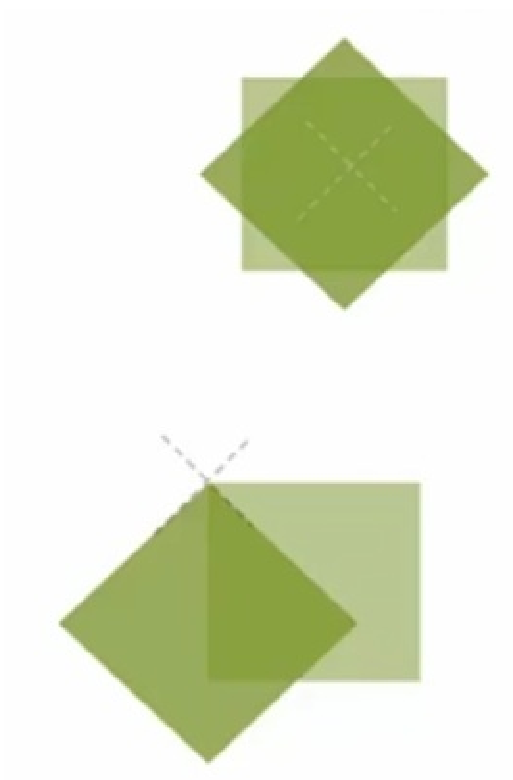
transform-origin

其用于设置原点的位置（默认位置为元素中心）第一值为 X 方向，第二值为 Y 方向， 第三值为 Z 方向。（当值空出未写的情况下默认为 50%）

```
transform-origin: [ <percentage> | <length> | left | center | ri
ght | top | bottom] | [ [ <percentage> | <length> | left | cente
r | right ] && [ <percentage> | <length> | top | center | bottom
 ] ] <length>?

<!-- 默认值为 50% 50% -->

transform-origin: 50% 50%;
transform-origin: 0 0;
transform-origin: right 50px 20px;
transform-origin: top right 20px;
```



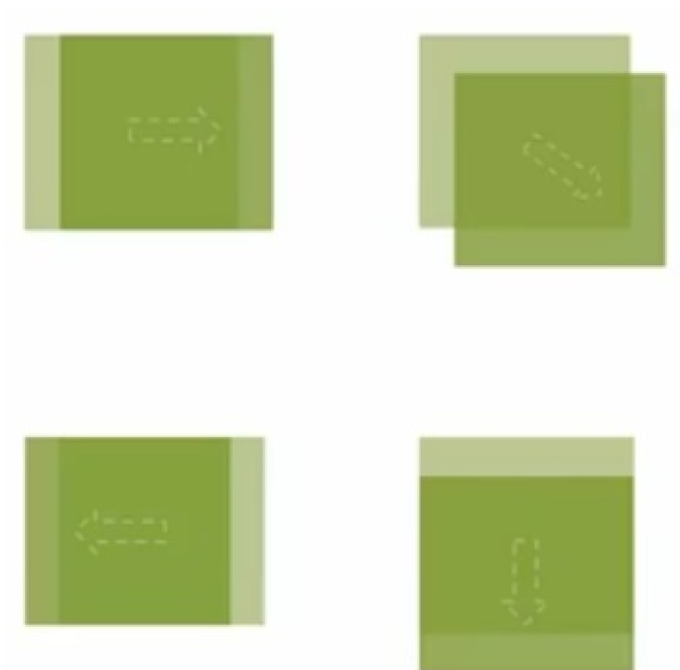
translate()

移动方法，参数分别代表 X 与 Y 轴的移动（偏移值均可为负值）。

```
translate(<translation-value>[, <translation-value>]?)

<!-- 也可单独设置 X 与 Y 轴的偏移 -->
translationX(<translation-value>)
translationY(<translation-value>)

transform: translate(50px);
transform: translate(50px, 20%);
<!-- Y 轴偏移为偏移对象的高度，X 轴为宽度 -->
transform: translate(-50px);
transform: translate(20%);
```



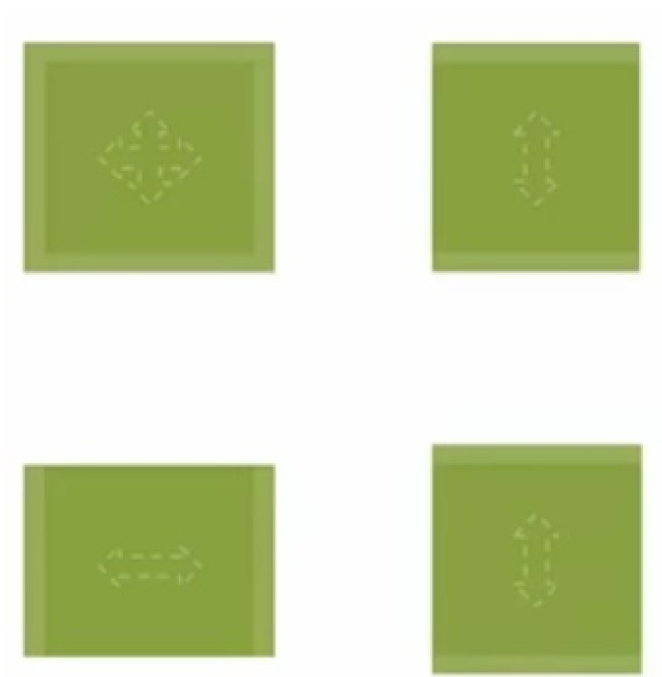
scale()

缩放方法，参数分别代表 X 与 Y 轴的缩放（缩放值均可为小数）。当第二值忽略时，默认设置为等同第一值。

```
scale(<number> [, <number>]?)

scaleX(<number>)
scaleY(<number>)

<!-- 整体放大 1.2 倍 -->
transform: scale(1.2);
<!-- 高度拉伸 -->
transform: scale(1, 1.2);
<!-- 宽度拉伸 -->
transform: scaleX(1.2);
<!-- 高度拉伸 -->
transform: scaleY(1.2);
```



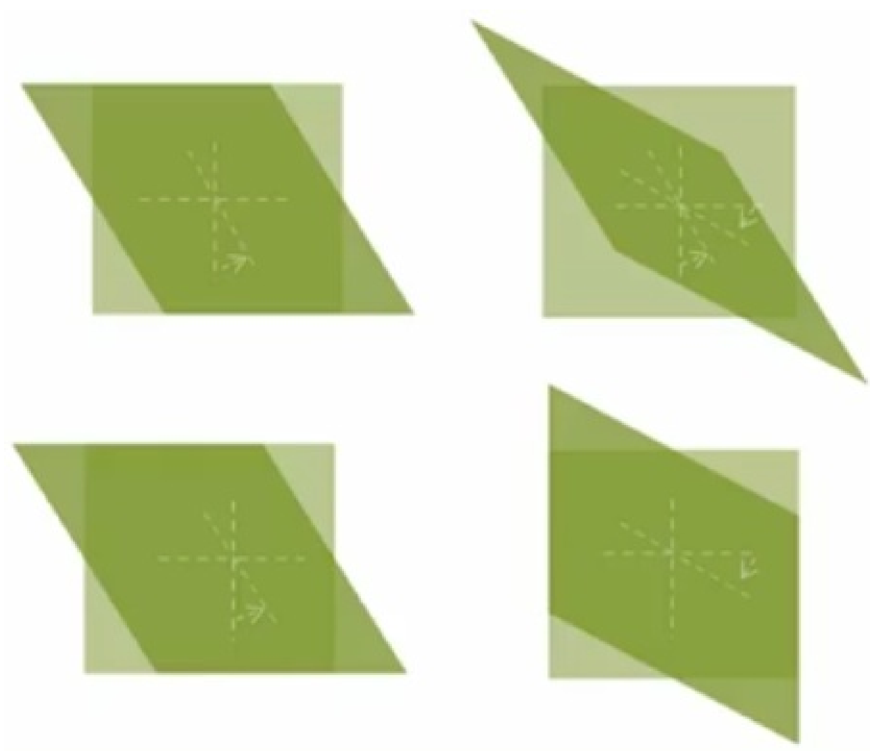
skew()

其为倾斜的方法。第一值为 Y 轴往 X 方向倾斜（逆时针），第二值为 X 轴往 Y 方向倾斜（顺时针）。（倾斜值可为负值）

```
skew(<angle>[, <angle>]?)

skewX(<angle>)
skewY(<angle>)

transform: skew(30deg);
transform: skew(30deg, 30deg);
transform: skewX(30deg);
transform: skewY(30deg);
```



3D 变形

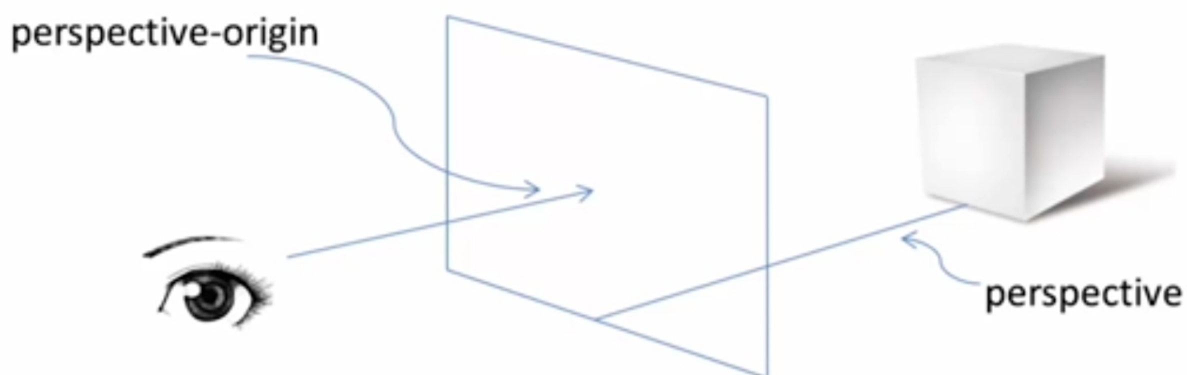
3D 变形示例代码

rotateY()

3D 空间旋转。

```
transform: rotateY(<angle>)
```

perspective



其用于设置图片 Y 轴旋转后的透视效果。`<length>` 可以理解为人眼与元素之间的距离，越紧则效果越明显。

```
perspective: none | <length>

perspective: none;
perspective: 2000px;
perspective: 500px;
```



perspective-origin

其为设定透视的角度（透视位置均可设定为负值）。

```
perspective-origin: [ <percentage> | <length> | left | center |
right | top | bottom] | [ [ <percentage> | <length> | left | cen
ter | right ] && [ <percentage> | <length> | top | center | bott
om ] ]

perspective-origin: 50% 50%
<!-- 默认值为 50% 50% 正中间的位置进行透视-->
perspective-origin: left bottom;
perspective-origin: 50% -800px;
perspective-origin: right;
```

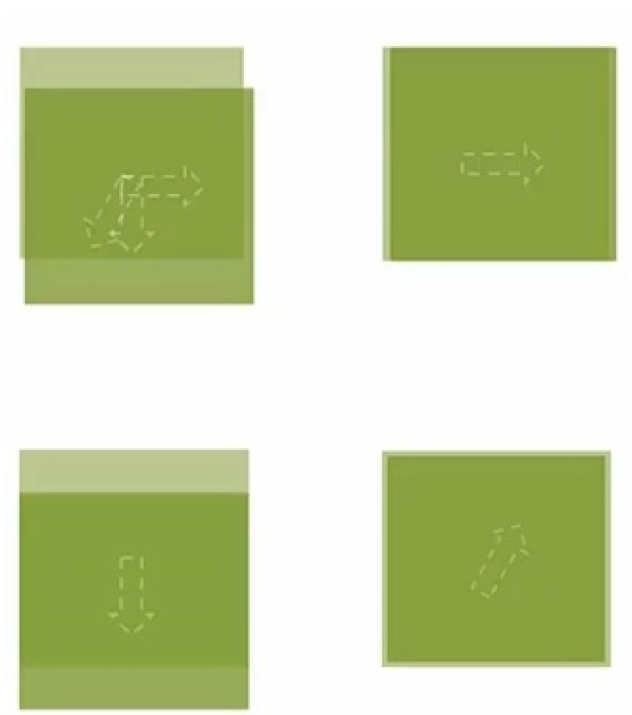


translate3d()

```
translate3d(<translate-value>, <translate-value>, <length>)

translateX(<translate-value>)
translateY(<translate-value>)
translateZ(<length>)

transform: translate3d(10px, 20%, 50px);
<!-- %的参照物为自身元素 -->
transform: translateX(10px);
transform: translateY(20%);
transform: translateZ(-100px);
```



scale3d()

```
scale3d(<number>, <number>, <number>)

scaleX(<number>)
scaleY(<number>)
scaleZ(<number>)

transform: scale3d(1.2, 1.2, 1);
transform: scale3d(1, 1.2, 1);
transform: scale3d(1.2, 1, 1);
transform: scaleZ(5);
<!-- Z 轴的缩放扩大并不影响盒子大小 -->
```



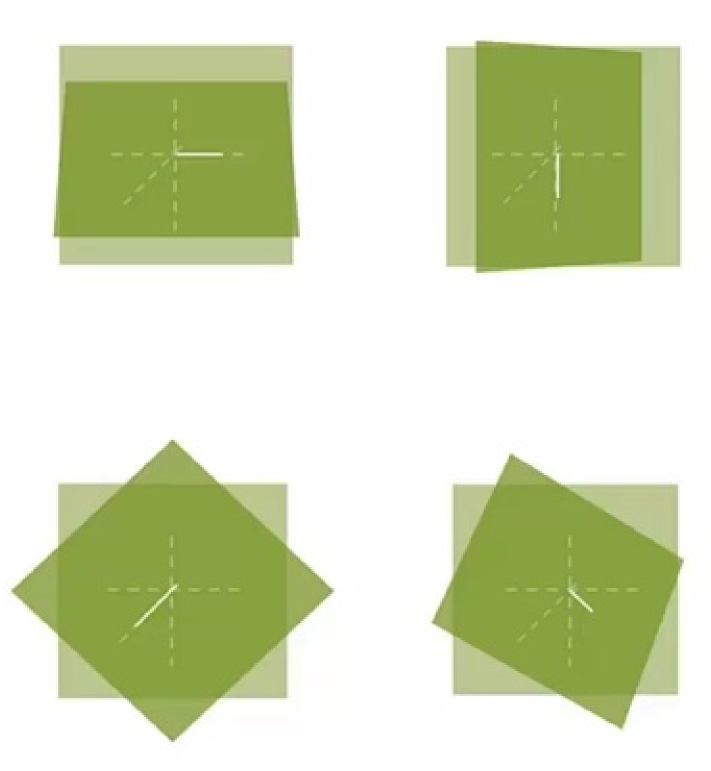
rotate3d()

取 X Y Z 三轴上的一点并于坐标原点连线，以连线为轴进行旋转（逆时针）。

```
rotate3d(<number>, <number>, <number>, <angle>)

rotateX(<angle>)
rotateY(<angle>)
rotateZ(<angle>)

transform: rotate3d(1, 0, 0, 45deg);
<!-- 上面等同于 X 轴旋转 -->
transform: rotate3d(0, 1, 0, 45deg);
<!-- 上面等同于 Y 轴旋转 -->
transform: rotate3d(0, 0, 1, 45deg);
<!-- 上面等同于 2D 旋转 -->
transform: rotate3d(1, 1, 1, 45deg);
```



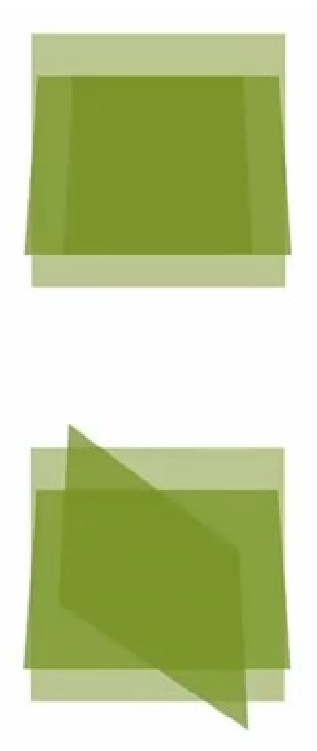
transform-style

其用于设置保留内部的 3D 空间，原因是一个元素进行 `transform` 之后内部默认为 `flat` 。

```
transform-style: flat | perserve-3d

<!-- 默认为 flat -->

transform-style: flat;
transform-style: preserve-3d;
```



backface-visibility

其用于设置背面不可见。

```
backface-visibility: visible | hidden

backface-visibility: visible;
backface-visibility: hidden;
```

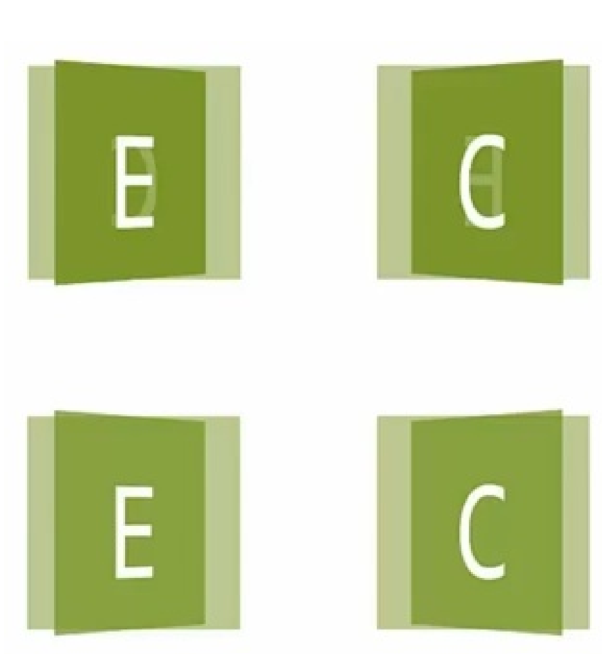


Table of Contents *generated with DocToc*

动画

动画示例代码

transition

过度动画

其为众多 `<single-transition>` 的值缩写。（当两个时间同时出现是，第一个时间为动画长度，第二个时间为动画延时）

```
transition: <single-transition> [',' <single-transition>]*

<single-transition> = [none | <single-transition-property>] || <
time> || <single-transition-timing-function> || <time>


transition: none;
transition: left 2s ease 1s, color 2s;
transition: 2s;
```

transition-property

```
transition-property: none | <single-traisition-property> [ ',' <
single-transition-property>]*

<single-transition-property> = all | <IDENT>

transition-property: none;
<!-- 默认值为 none -->
transition-property: all;
transition-property: left;
transition-property: left, color;
```

transition-duration

```
transition-duration: <time>[, <time>]*

transition-duration: 0s;
transition-duration: 1s;
transition-duration: 1s, 2s, 3s;
```

transition-delay

```
transition-delay: <time>[,<time>]*

transition-delay: 0s;
transition-delay: 1s;
transition-delay: 1s, 2s, 3s;
```

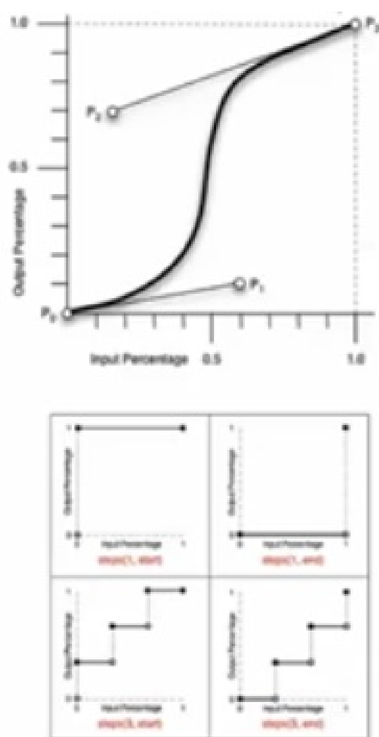
transition-timing-function

```
transition-timing-function: <single-transition-timing-function>[
',' <single-transition-timing-function>]*

<!-- 默认函数为 ease -->
<single-transition-timing-function> = ease | linear | ease-in |
ease-out | ease-in-out | cubic-bezier(<number>,<number>,<number>
,<number>) | step-start | step-end | steps(<integer>)[, [start |
 end]]?)

<!-- 对于 cubic-bezier 的曲线，前两个值为 P1 的坐标，后两值为 P2 的坐标
 -->

transition-timing-function: ease;
transition-timing-function: cubic-bezier(0.25, 0.1, 0.25, 1);
transition-timing-function: linear;
transition-timing-function: cubic-bezier(0, 0, 1, 1);
```



animation

```
animation: <single-animation> [',' <single-animation>]*

<single-animation> = <single-animation-name> || <time> || <single-animation-timing-function> || <time> || <single-animation-iteration-count> || <single-animation-direction> || <single-animation-fill-mode> || single-animation-play-state>

animation: none;
animation: abc 2s ease 0s 1 normal none running;
animation: abc 2s;
animation: abc 1s 2s both, abcd 2s both;
<!-- 调用多个动画 -->
```

动画可自动运行，但 `transition` 需要触发。

animation-name

`animation-name` 的名字可自由定义。

```
animation-name: <single-animation-name>#

<single-animation-name> = none | <IDENT>

animation-name: none;
animation-name: abc;
animation-name: abc, abcd;
```

animation-duration

与 `transition-duration` 属性值类似。

```
animation-duration: <time>[, <time>]*

animation-duration: 0s;
animation-duration: 1s;
animation-duration: 1s, 2s, 3s;
```

animation-timing-function

其与之前的 `transition-timing-function` 完全一模一样。

```
animation-timing-function: <timing-function>#

<single-timing-function> = <single-transition-timing-function>

animation-timing-function: ease;
animation-timing-function: cubic-bezier(0.25, 0.1, 0.25, 1);
animation-timing-function: linear;
animation-timing-function: cubic-bezier(0, 0, 1, 1);
animation-timing-function: ease, linear;
```

animation-iteration-count

其用于动画执行的次数（其默认值为 1）。

```
animation-iteration-count: <single-animation-iteration-count>#

<single-animation-iteration-count> = infinite | <number>

animation-iteration-count: 1;
animation-iteration-count: infinite;
animation-iteration-count: 1, 2, infinite;
```

animation-direction

其用于定义动画的运动方向。

```
animation-direction:<single-animation-direction>#

<single-animation-direction> = normal | reverse | alternate | al
ternate-revers

animation-direction: reverse
<!-- 动画相反帧的播放 -->
animation-direction: alternate
<!-- 往返执行动画 -->
animation-direction: alternate-revers
<!-- 相反的往返动画 -->
```

animation-play-state

其用于设定动画的播放状态。

```
animation-play-state: <single-animation-play-state>#

<single-animation-play-state> = running | paused

animation-play-state: running;
animation-play-state: pasued;
animation-play-state: running, paused;
```

animation-delay

其用于设置动画的延时，同 `transition-delay` 值相同。

```
animation-delay: <time>[, <time>]*
anim
animation-delay: 0s;
animation-delay: 1s;
animation-delay: 1s, 2s, 3s;
```

animation-fill-mode

其用于设置动画开始时，是否保持第一帧的动画和动画在结束时时候保持最后的状态。

```
animation-fill-mode: <single-animation-fill-mode>[',' <single-an
imation-fill-mode>]*

<single-animation-fill-mode> = none | backwards | forwards | bot
h

animation-fill-mode: none;
<!-- 不做设置 -->
animation-fill-mode: backwards;
<!-- 动画开始时出现在第一帧的状态 -->
animation-fill-mode: forwards;
<!-- 动画结束时保留动画结束时的状态 -->
animation-fill-mode: both;
<!-- 开始和结束时都应保留关键帧定义的状态（通常设定） -->
animation-fill-mode: forwards, backwards;
```

@keyframes

其用于定义关键帧。

```
<!-- 写法一 -->
@keyframes abc {
  from {opacity: 1; height: 100px;}
  to {opacity: 0.5; height: 200px;}
}

<!-- 写法二 -->
@keyframes abcd {
  0% {opacity: 1; height: 100px;}
  100% {opacity: 0.5; height: 200px}
}

@keyframes flash {
  0%, 50%, 100% {opacity: 1;}
  25%, 75% {opacity: 0;}
}

<!-- 例子 -->
animation: abc 0.5s both;
animation: flash 0.5s both;
animaiton: abc 0.5s both, flash 0.5s both;
```



常见布局样例

自动居中一列布局

所需知识：

- 标准文档流
- 块级元素
- margin 外边距属性

```
<style type="text/css" media="screen">
  article {
    width: 800px;
    margin: 0 auto;
  }
</style>

<body>
  <article>
    <h1>Title</h1>
    <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit.
 A natus repellendus, modi culpa, dolor ducimus debitis, facere
dolorum cum aspernatur, soluta molestias est illo vel iusto esse
 ex ullam amet!</p>
  </article>
</body>
```

NOTE：设置 auto 会根据浏览器宽度自动设置外边距尺寸。在设置浮动或绝对定位则会使自动居中失效，因为其会脱离文档流。

`(浏览器宽度 - 外包含层的宽度)/ 2 = 外边距` 。

横向两列布局

此方法也同时可以实现横向多列布局（原理与两列布局相同）。

所需知识：

- float 属性，使纵向排列的块级元素，横向排列

- margin 属性，设置列之间的间距

```
<style type="text/css" media="screen">
  .clearfix:after {
     content: "."; /* Older browser do not support empty content
 */
     visibility: hidden;
     display: block;
     height: 0;
     clear: both;
  }
  .clearfix {zoom: 1;} /* 针对 IE 不支持 :after */
  body {
    width: 930px;
    margin: 0 auto; /* 横向居中 */
  }
  article {
    width: 800px;
    float: left;
    margin-right: 10px;
  }
  aside {
    width: 120px;
    float: right;
  }
</style>
<body class="clearfix">
  <article>
    <h1>Title</h1>
    <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit.
 Explicabo, quam, fugit. Accusamus voluptates nesciunt in, autem
 ipsa assumenda a iusto, reiciendis earum repudiandae, nulla nat
us blanditiis, aliquam asperiores commodi qui.</p>
  </article>
  <aside>
    <h3>Aside Title</h3>
    <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit.
 Expedita, molestiae!</p>
  </aside>
</body>
```

绝对定位的横向两列布局

应用场合较少，常用与一列定宽，另一列自适应。

需要知识：

- relative positon 父元素相对定位
- absolute 自适应宽度元素绝对定位

注意：固定宽度列的高度需大于自适应的列（原因是绝对定位会脱离文档流，不能撑开父元素）。

```
<style type="text/css" media="screen">
  body {
    position: relative;
    width: 100%;
  }

  article {
    position: absolute;
    top: 0;
    right: 0;
    width: 800px;
  }
  aside {
    position: absolute;
    top: 0;
    right:800px;
    left: 0;
  }
</style>

<body>
  <article>
    <h1>Title</h1>
    <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit.
 Error obcaecati sint minima totam fuga, tempora, id quia soluta
 officia iure eligendi sequi non dicta, doloribus accusamus odit
 fugiat quam quibusdam.</p>
  </article>
  <aside>
    <h3>Aside Title</h3>
    <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit.
 Atque, doloremque.</p>
  </aside>
</body>
```

JavaScript 程序设计

Javascript 程序设计以 ECMAScript 5.1 为标准，从基本语法到原理深入,理解和编写Javascript程序。核心内容有语言简介、调试器、类型系统、内置对象、基本语法、变量作用域、闭包、面向对象编程等。

JavaScript 介绍

前端开发三要素，`HTML`（描述网页内容），`CSS`（描述样式），`JavaScript`（控制网页行为）。JavaScript 为解释型编程语（程序不需要编译，程序在运行时才翻译成机器语言，每执 行一次都要翻译一次)，运行环境也很广泛（浏览器或操作系统中 NodeJS）。

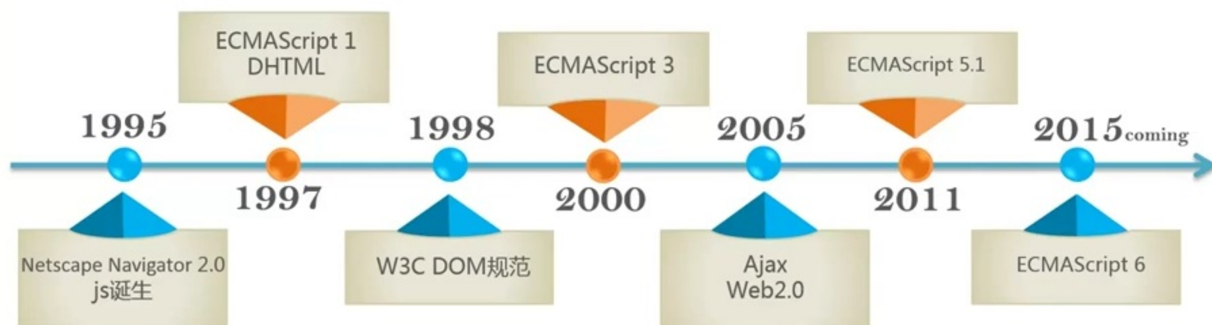
JavaScript 又分为两部分，ECMAScript 和 DOM。其中 JavaScript 的基本的语言属性（例如，变量，作用域等）都在 ECMAScript 中有详细的定义好比汉子在字典中的定义。DOM 中则定义如何修改文档结构（HTML 或 XML 等）。

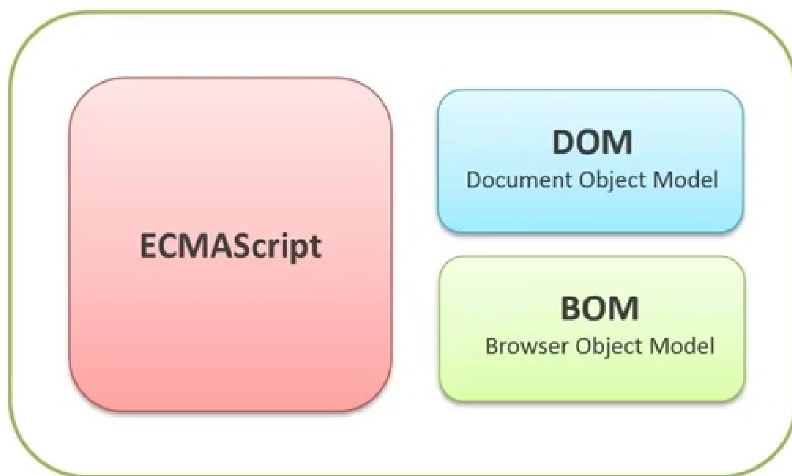
JavaScript的引入方法如下：

```
<!DOCTYPE html>
<html>
<head>
  <title>Demo Page</title>
</head>
<body>
  <!-- 以上代码忽略 -->

  <!-- 为性能优化需将 javascript 代码放置在 body 标签的最末端 -->
  <!-- 外联文件 -->
  <script src="/javascripts/application.js" type="text/javascrip
t" charset="utf-8" async defer></script>
  <!-- 内嵌代码 -->
  <script>

  </script>
</body>
</html>
```





ECMAScript

- js语言核心标准

DOM文档对象模型

- W3C 标准
- 操作文档对象的api

BOM 浏览器对象模型

- 操作浏览器对象的api

调试器

调试工具都内置于主流浏览器中（Firefox 中需独立下载 Firebug）。更多关于 Google Chrome DevTools 的信息可以在这里找到。

基本语法

变量标示符

变量的命名

```
var _name = null;
var $name = null;
var name0 = null;
```

关键字与保留字

JavaScript 在语言定义中保留的字段，这些字段在语言使用中存在特殊意义或功能，在程序编写的过程中不可以当做变量或函数名称使用。无需记忆，报错修改即可。

关键字与保留字可以在 Mozilla 开发者站点找到。

字符敏感

字符串的大小写是有所区分的，不同字符指代不同的变量。

严格模式

增益

- 消除语法中不合理与不安全的问题，保证代码正常运行
- 提高编译效率，增加运行速度

使用方法

```
<!-- 全局使用 严格 模式 -->
"use strict";
(function(){
  console.log('>>> Hello, world!');
})()

<!-- 或者在函数内部声明使用 严格 模式 -->
(function(){
  "use strict";
  console.log('>>> Hello, world!');
})()
```

严格模式与标准模式的区别：

- 严格模式下隐式声明或定义变量被静止
- 严格模式下对象重名的属性在严格模式下被静止
- 严格模式下 `arguments.callee()` 被禁用
- 严格模式下 `with()` 语句
- 更多限制

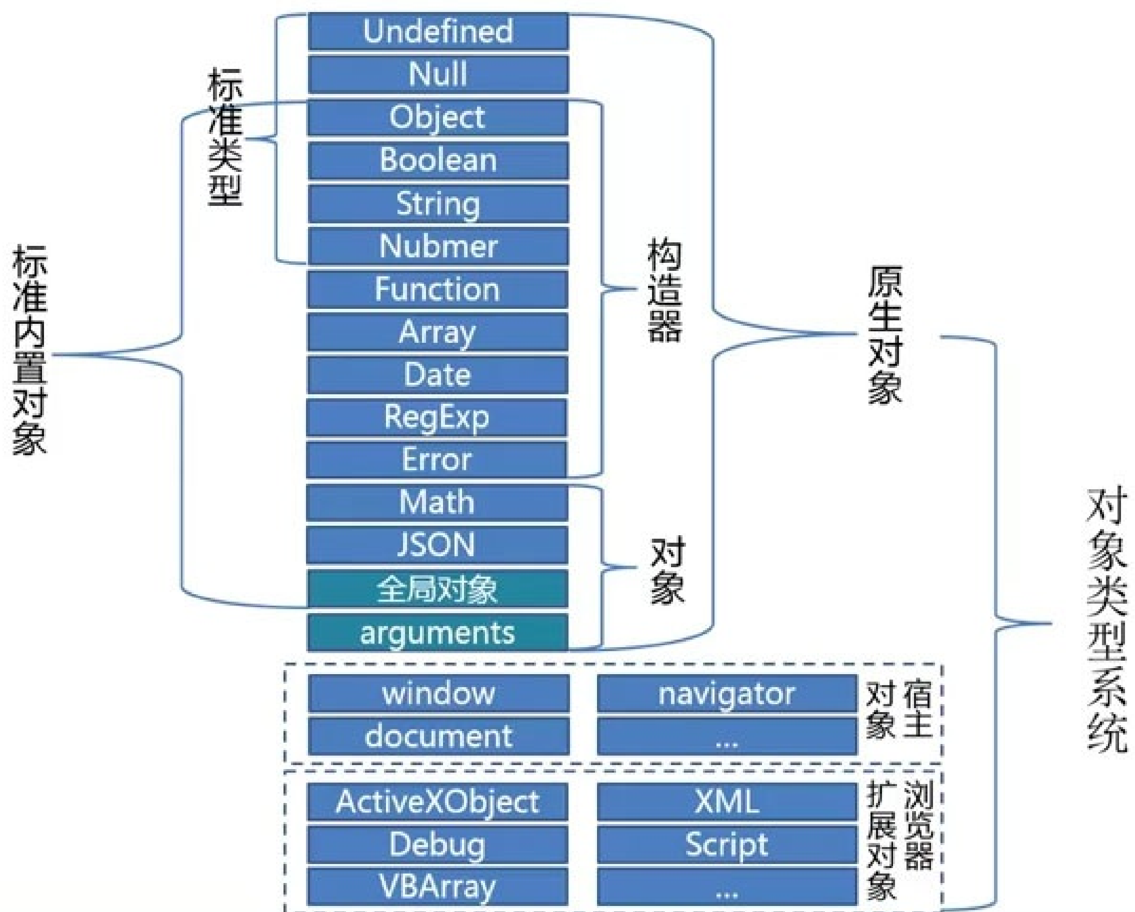
注释

```
/*
  多行注释，不可嵌套
 */

// 单行注释
```

类型系统

javascript 类型系统可以分为标准类型和对象类型，进一步标准类型又可以分为原始类型和引用类型，而对象类型又可以分为内置对象类型、普通对象类型、自定义对象类型。



标准类型

标准类型共包括了**6**个分别是：

原始类型（值类型）：

- Undefined `undefined`
- Null `null`
- Boolean `true`
- String `'hello'`
- Number `123`

引用类型（对象类型）：

- Object

```
var obj = {};
<!-- 原始类型变量的包装类型如下 -->
var bool = new Boolean(true);
var str = new String("hello");
var num = new Number(1);
var obj0 = new Object();
```

原始类型和引用类型的区别：

原始类型储存在栈（Stack）中储存变量的值，而引用类型在栈中保存的是所引用内容储存在堆（Heap）中的值。类似于指针的概念，引用类型并非储存变量真实数值而是地址，所以对已引用类型的复制其实只是复制了相同的地址而非实际的变量值。

Undefined 值：undefined 出现场景：

- 以声明为赋值的变量 `var obj;`
- 获取对象不存在的属性 `var obj = {x: 0}; obj.y;`
- 无返回值函数的执行结果 `function f(){}; var obj = f();`
- 函数参数没有传入 `function f(i){console.log(i)}; f();`
- `void(expression)`

Null 值：null 出现场景：

- 获取不存在的对象 `document.getElementById('not-exist-element')`

Boolean 值：true, false 出现场景：

- 条件语句导致的系统执行的隐式类型转换 `if(隐式转换){}`
- 字面量或变量定义 `var bool = true;`

String 值：字符串 出现场景：

- `var str = 'Hello, world!';`

Number 值：整型直接量，八进制直接量（0-），十六进制直接量（0x-），浮点型直接量 出现场景：

- `1026`
- `3.14`
- `1.2e5`
- `0x10`

Object 值：属性集合 出现场景：

- `var obj = {name: 'Xinyang'};`

变量转换表

Value	Boolean	Number	String
undefined	false	NaN	"undefined"
null	false	0	"null"
true	true	1	"true"
false	false	0	"false"
''	false	0	''
'123'	true	123	'123'
'1a'	true	NaN	'1a'
0	false	0	"0"
1	true	1	"1"
Infinity	true	Infinity	"Infinity"
NaN	false	NaN	'NaN'
{}	true	NaN	"[object Object]"

类型识别

- `typeof`
- `Object.prototype.toString`
- `constructor`
- `instanceof`

typeof：

- 可以是标准类型（Null 除外）

- 不可识别具体的对象类型（Function 除外）

Object.prototype.toString：

- 可是识别标准类型及内置对象类型（例如，Object, Date, Array）
- 不能识别自定义对象类型

constructor：

- 可以识别标准类型（Undefined/Null 除外）
- 可识别内置对象类型
- 可识别自定义对象类型

```
function getConstructiorName(obj) {
  return obj && obj.constructor && obj.constructor.toString().match(/function\s*([^(]*)/)[1];
}
getConstructiorName([]) === "Array"; // true
```

instanceof：

- 不可判别原始类型
- 可判别内置对象类型
- 可判别自定义对象类型

Table of Contents *generated with DocToc*

类型识别

- `typeof`
- `Object.prototype.toString`
- `constructor`
- `instanceof`

typeof：

- 可以是标准类型（Null 除外）
- 不可识别具体的对象类型（Function 除外）

Object.prototype.toString：

- 可是识别标准类型及内置对象类型（例如，Object, Date, Array）
- 不能识别自定义对象类型

constructor：

- 可以识别标准类型（Undefined/Null 除外）
- 可识别内置对象类型
- 可识别自定义对象类型

```
function getConstructiorName(obj) {
  return obj && obj.constructor && obj.constructor.toString().match(/function\s*([^(]*)/)[1];
}
getConstructiorName([]) === "Array"; // true
```

instanceof：

- 不可判别原始类型
- 可判别内置对象类型
- 可判别自定义对象类型

JavaScript的数据类型可以分为：标准类型和对象类型。

`标准类型有`：undefined Null Boolean Date Number Object

`对象类型（构造器类型）`：Boolean Date Number Object Array Date Error Function RegExp

用来判断数据类型的一般有四种方式，分别是：

- `typeof`
- `Prototype.toString()`
- `constructor`
- `instanceof`

下面我们写一个**HTML**来检验一下：

```
<html>
<head>
    <title>JavaScript类型判断</title>
    <meta charset="utf-8">
    <style type="text/css">
        .red{
            background-color:red;
        }
    </style>
</head>
<body>
    <script type="text/javascript">
        /* Standard Type */
        var a;    //undefined
        var b = document.getElementById("no_exist_element"); //n
ull
        var c = true;    //Boolean
        var d = 1;    //Number
        var e = "str";    //String
        var f = {name : "Tom"};    //Object

        /* Object Type */
        var g = new Boolean(true);    //Boolean Object
        var h = new Number(1);    //Number Object
        var i = new String("str");    //String Object
        var j = new Object({name : "Tom"}); //Object Object
        var k = new Array([1, 2, 3, 4]);    //Array Object
```

```
        var l = new Date();    //Date Object
        var m = new Error();
        var n = new Function();
        var o = new RegExp("\\d");

        /* Self-Defined Object Type */
        function Point(x, y) {
          this.x = x;
          this.y = y;
        }

        Point.prototype.move = function(x, y) {
          this.x += x;
          this.y += y;
        }

        var p = new Point(1, 2);

        /* Use the Prototype.toString() to judge the type */
        function type(obj){
            return Object.prototype.toString.call(obj).slice(8,
-1).toLowerCase();
        }
    </script>
    <table border="1" cellspacing="0">
        <tr>
            <td></td>
            <td>typeof</td>
            <td>toString</td>
            <td>constructor</td>
            <td>instanceof</td>
        </tr>
        <tr>
            <td>undefined</td>
            <td><script type="text/javascript">document.write(ty
peof a)</script></td>
            <td><script type="text/javascript">document.write(ty
pe(a))</script></td>
            <td class="red"><script type="text/javascript">docum
ent.write(a.constructor)</script></td>
```

```
            <td class="red"><script type="text/javascript">docum
ent.write(a instanceof "undefined")</script></td>
        </tr>
        <tr>
            <td>Null</td>
            <td class="red"><script type="text/javascript">docum
ent.write(typeof b);</script></td>
            <td><script type="text/javascript">document.write(ty
pe(b));</script></td>
            <td class="red"><script type="text/javascript">docum
ent.write(b.constructor);</script></td>
            <td class="red"><script type="text/javascript">docum
ent.write(b instanceof "null");</script></td>
        </tr>
        <tr>
            <td>Boolean</td>
            <td><script type="text/javascript">document.write(ty
peof c);</script></td>
            <td><script type="text/javascript">document.write(ty
pe(c));</script></td>
            <td><script type="text/javascript">document.write(c.
constructor);</script></td>
            <td class="red"><script type="text/javascript">docum
ent.write(c instanceof "boolean");</script></td>
        </tr>
        <tr>
            <td>Number</td>
            <td><script type="text/javascript">document.write(ty
peof d);</script></td>
            <td><script type="text/javascript">document.write(ty
pe(d));</script></td>
            <td><script type="text/javascript">document.write(d.
constructor);</script></td>
            <td class="red"><script type="text/javascript">docum
ent.write(d instanceof "number");</script></td>
        </tr>
        <tr>
            <td>String</td>
            <td><script type="text/javascript">document.write(ty
peof e);</script></td>
```

```
            <td><script type="text/javascript">document.write(ty
pe(e));</script></td>
            <td><script type="text/javascript">document.write(e.
constructor);</script></td>
            <td class="red"><script type="text/javascript">docum
ent.write(e instanceof "string");</script></td>
        </tr>
        <tr>
            <td>Object</td>
            <td><script type="text/javascript">document.write(ty
peof f);</script></td>
            <td><script type="text/javascript">document.write(ty
pe(f));</script></td>
            <td><script type="text/javascript">document.write(f.
constructor);</script></td>
            <td class="red"><script type="text/javascript">docum
ent.write(f instanceof "object");</script></td>
        </tr>
        <tr><td colspan="5">-----------------------</td></tr>
        <tr>
            <td>Boolean Object</td>
            <td class="red"><script type="text/javascript">docum
ent.write(typeof g);</script></td>
            <td><script type="text/javascript">document.write(ty
pe(g));</script></td>
            <td><script type="text/javascript">document.write(g.
constructor);</script></td>
            <td><script type="text/javascript">document.write(g
instanceof Boolean);</script></td>
        </tr>
        <tr>
            <td>Number Object</td>
            <td class="red"><script type="text/javascript">docum
ent.write(typeof h);</script></td>
            <td><script type="text/javascript">document.write(ty
pe(h));</script></td>
            <td><script type="text/javascript">document.write(h.
constructor);</script></td>
            <td><script type="text/javascript">document.write(h
instanceof Number);</script></td>
```

```
        </tr>
        <tr>
            <td>String Object</td>
            <td class="red"><script type="text/javascript">docum
ent.write(typeof i);</script></td>
            <td><script type="text/javascript">document.write(ty
pe(i));</script></td>
            <td><script type="text/javascript">document.write(i.
constructor);</script></td>
            <td><script type="text/javascript">document.write(i
instanceof String);</script></td>
        </tr>
        <tr>
            <td>Object Object</td>
            <td class="red"><script type="text/javascript">docum
ent.write(typeof j);</script></td>
            <td><script type="text/javascript">document.write(ty
pe(j));</script></td>
            <td><script type="text/javascript">document.write(j.
constructor);</script></td>
            <td><script type="text/javascript">document.write(j
instanceof Object);</script></td>
        </tr>
        <tr>
            <td>Array Object</td>
            <td class="red"><script type="text/javascript">docum
ent.write(typeof k);</script></td>
            <td><script type="text/javascript">document.write(ty
pe(k));</script></td>
            <td><script type="text/javascript">document.write(k.
constructor);</script></td>
            <td><script type="text/javascript">document.write(k
instanceof Array);</script></td>
        </tr>
        <tr>
            <td>Date Object</td>
            <td class="red"><script type="text/javascript">docum
ent.write(typeof l);</script></td>
            <td><script type="text/javascript">document.write(ty
pe(l));</script></td>
```

```
            <td><script type="text/javascript">document.write(l.
constructor);</script></td>
            <td><script type="text/javascript">document.write(l
instanceof Date);</script></td>
        </tr>
        <tr>
            <td>Error Object</td>
            <td class="red"><script type="text/javascript">docum
ent.write(typeof m);</script></td>
            <td><script type="text/javascript">document.write(ty
pe(m));</script></td>
            <td><script type="text/javascript">document.write(m.
constructor);</script></td>
            <td><script type="text/javascript">document.write(m
instanceof Error);</script></td>
        </tr>
        <tr>
            <td>Function Object</td>
            <td><script type="text/javascript">document.write(ty
peof n);</script></td>
            <td><script type="text/javascript">document.write(ty
pe(n));</script></td>
            <td><script type="text/javascript">document.write(n.
constructor);</script></td>
            <td><script type="text/javascript">document.write(n
instanceof Function);</script></td>
        </tr>
        <tr>
            <td>RegExp Object</td>
            <td class="red"><script type="text/javascript">docum
ent.write(typeof o);</script></td>
            <td><script type="text/javascript">document.write(ty
pe(o));</script></td>
            <td><script type="text/javascript">document.write(o.
constructor);</script></td>
            <td><script type="text/javascript">document.write(o
instanceof RegExp);</script></td>
        </tr>
        <tr><td colspan="5">-----------------------</td></tr>
        <tr>
```

```
                <td>Point Objct</td>
                <td class="red"><script type="text/javascript">document.write(typeof p);</script></td>
                <td class="red"><script type="text/javascript">document.write(type(p));</script></td>
                <td><script type="text/javascript">document.write(p.constructor);</script></td>
                <td><script type="text/javascript">document.write(p instanceof Point);</script></td>
            </tr>
        </table>
</body>
</html>
```

执行的结果如下：

	typeof	toString	constructor	instanceof
undefined	undefined	undefined		
Null	object	null		
Boolean	boolean	boolean	function Boolean() { [native code] }	
Number	number	number	function Number() { [native code] }	
String	string	string	function String() { [native code] }	
Object	object	object	function Object() { [native code] }	

Boolean Object	object	boolean	function Boolean() { [native code] }	true
Number Object	object	number	function Number() { [native code] }	true
String Object	object	string	function String() { [native code] }	true
Object Object	object	object	function Object() { [native code] }	true
Array Object	object	array	function Array() { [native code] }	true
Date Object	object	date	function Date() { [native code] }	true
Error Object	object	error	function Error() { [native code] }	true
Function Object	function	function	function Function() { [native code] }	true
RegExp Object	object	regexp	function RegExp() { [native code] }	true
----------------------				
Point Objct	object	object	function Point(x, y) { this.x = x; this.y = y; }	true

其中红色的单元格表示该判断方式不支持的类型。

Table of Contents *generated with DocToc*

内置对象

通常情况下只有对象才存在方法，但 JavaScript 不同它具有12种内置对象。内置对象又分为两类，普通对象（属性和方法）与构造器对象（可用于实例化普通对象，它还包含原型对象属性和方法，及实例对象属性和方法）。

JavaScript 对象原型链的简要说明

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
Point.prototype.move = function(x, y) {
  this.x += x;
  this.y += y;
}
var p = new Point(1, 1);
p.move(2,2);
```

`__proto__` 称之为原型链，有如下特点：

1. `__proto__` 为对象内部的隐藏属性
2. `__proto__` 为实例化该对象的构造器的 `prototype` 对象的引用，因此可以直接方法 `prototype` 的所有属性和方法
3. 除了 `Object` 每个对象都有一个 `__proto__` 属性且逐级增长形成一个链，原型链顶端是一个 `Object` 对象。
4. 在调用属性或方法时，引擎会查找自身的属性如果没有则会继续沿着原型链逐级向上查找，直到找到该方法并调用。
5. `__proto__` 跟浏览器引擎实现相关，不同的引擎中名字和实现不尽相同

(chrome、firefox中名称是 `__proto__` ，并且可以被访问到，IE中无法访问)。基于代码兼容性、可读性等方面的考虑，不建议开发者显式访问 `__proto__` 属性或通过 `__proto__` 更改原型链上的属性和方法，可以通过更改构造器 `prototype` 对象来更改对象的 `__proto__` 属性。

构造器对象与普通对象的区别

```
> function Point(x, y) {this.x = x; this.y=y;}
<· undefined
> Point.prototype.move = function(x, y) {this.x += x; this.y+=y;}
<· function Point.move(x, y)
> var point = new Point(1, 2);
<· undefined
> point
<· ▼Point {x: 1, y: 2} i
      x: 1
      y: 2
    ▼__proto__: Point
      ▶constructor: function Point(x, y) {this.x = x; this.y=y;}
      ▶move: function (x, y) {this.x += x; this.y+=y;}
      ▼__proto__: Object
        ▶__defineGetter__: function __defineGetter__() { [native code] }
        ▶__defineSetter__: function __defineSetter__() { [native code] }
        ▶__lookupGetter__: function __lookupGetter__() { [native code] }
        ▶__lookupSetter__: function __lookupSetter__() { [native code] }
        ▶constructor: function Object() { [native code] }
        ▶hasOwnProperty: function hasOwnProperty() { [native code] }
        ▶isPrototypeOf: function isPrototypeOf() { [native code] }
        ▶propertyIsEnumerable: function propertyIsEnumerable() { [native code…
        ▶toLocaleString: function toLocaleString() { [native code] }
        ▶toString: function toString() { [native code] }
        ▶valueOf: function valueOf() { [native code] }
        ▶get __proto__: function __proto__() { [native code] }
        ▶set __proto__: function __proto__() { [native code] }
```

1. 构造器对象原型链中的 `__proto__` 是一个 `Function.prototype` 对象的引用，因此可以调用 `Function.prototype` 的属性及方法
2. 构造器对象本身有一个 `prototype` 属性，用该构造器实例化对象时该 `prototype` 会被实例对象的 `__proto__` 所引用
3. 构造器对象本身是一个 `function` 对象，因此也会有自身属性

标准内置对象

构造器对象

- Object

- Boolean
- String
- Number
- Function
- Array
- RegExp
- Date
- Error

其他对象

- Math
- JSON
- 全局对象

内置对象，其实也叫内置构造器，它们可以通过 `new` 的方式创建一个新的实例对象。内置对象所属的类型就叫内置对象类型。其声明方式如下：

```
var i = new String("str");          // String Object
var h = new Number(1);              // Number Object
var g = new Boolean(true);          // Boolean Object
var j = new Object({name : "Tom"}); // Object Object
var k = new Array([1, 2, 3, 4]);    // Array Object
var l = new Date();                 // Date Object
var m = new Error();
var n = new Function();
var o = new RegExp("\\d");
```

注意：虽然标准类型中有 `Boolean` `String` `Number` `Object` ，内置对象类型中也有 `Boolean` `String` `Number` `Object` ，但它们其实是通过不同的声明方式来进行区别的。标准类型通过直接赋值，而对象类型则是通过构造器实现初始化。

Object

> 构造器的原型对象在对象实例化时将会被添加到实例对象的原型链当中。
> `__proto__` 为原型链属性，编码时不可被显像调用。但是实例化对象可以调用原型链上的方法。

用 String/Number 等构造器创建的对象原型链顶端对象始终是一个Object对象，因此这些对象可以调用Object的原型对象属性和方法。所以 String/Number 等构造器是 Object 的子类。

更多关于 Object 的内容可以在这里找到。

构造器说明：

- Object 是属性和方法的集合
- String/Number/Boolean/Array/Date/Error 构造器均为 Object 的子类并集成 Object 原型对象的属性及方法。

实例化方法

```
var obj0 = new Object({name: 'X', age: 13});
// 常用方法
var obj1 = {name: 'Q', age: 14};
```

属性及方法

- prototype
- create
- keys
- ...

**原型对象属性及其方法

- constructor
- toString
- valueOf
- hasOwnProperty
- ...

实例对象属性及方法

无

Object.create

功能：基于原型对象创造新对象

```
// Object.create(prototype[, propertiesObject])
var prototype = {name: 'X', age: 13};
var obj = Object.create(proto);
```

Object.prototype.toString

功能：获取方法调用者的标准类型

```
// objectInstance.toString()
var obj = {};
obj.toString(); // Object
```

Object.prototype.hasOwnProperty

功能：判断一个属性是否是一个对象的自身属性

```
// objectInstance.hasOwnProperty("propertyName")
var obj = Object.create({a: 1, b: 2});
obj.c = 3;
obj.hasOwnProperty('a'); // false
obj.hasOwnProperty('c'); // true
```

Boolean

构造器说明：值为 true 与 false

属性及方法

- prototype

**原型对象属性及其方法

- constructor, toString, valueOf

String

构造器说明：单双引号内的字符串

实例化方法

```
'Hello, world!'
var str0 = 'Xinyang';
var str1 = new String('Xinyang');
```

属性及方法

- prototype
- fromCharCode（转换 ASCII 代码为字符）

原型对象属性及其方法

- constructor
- indexOf
- replace
- slice
- split
- charCodeAt
- toLowerCase
- ...

String.prototype.indexOf

功能：获取子字符串在字符串中的索引

```
// stringObject.indexOf(searchValue, fromIndex)
var str = "I am X. From China!";
var index = str.indexOf('a'); // 2
str.indexOf('a', index + 1); // 16
str.indexOf('Stupid'); // -1 字符串不存在
```

String.prototype.replace

功能：查找字符串替换成目标文字

```
// stringObject.replace(regexp/substr, replacement)
var str = "apple is bad";
str = str.replace('bad', 'awesome');
```

String.prototype.split

功能：按分隔符将分隔符分成字符串数组

```
// stringObject.split(separator, arrayLength)
var str = '1 2 3 4';
str.split(' '); // ['1', '2', '3', '4'];
str.split(' ', 3); // ['1', '2', '3'];
str.split(/\d+/); // ["", " ", " ", " ", ""]
```

Number

构造器说明：整型直接量，八进制直接量（0-），十六进制直接量（0x-)，浮点型直接量

实例化方法

```
10
1.2e5
var count = 0x10;
var pi = new Number(3.1415);
```

属性及方法

- prototype
- MAX_VALUE
- MIN_VALUE
- NaN
- NEGATIVE_INFINITY
- POSITIVE_INFINITY

原型对象属性及其方法

- constructor
- toFixed
- toExponential
- ...

Number.prototype.toFixed

功能：四舍五入至指定小数位

```
// numberObject.toFixed(num)
var num0 = 3.14;
num0.toFixed(1); // 3.1
var num1 = 3.35;
num1.toFixed(1); // 3.4
```

Array

构造器说明：定义数组对象

实例化方法

```
var a0 = [1, 'abc', true, function(){}];
var a1 = new Array();
var a2 = new Array(1, 'abc', true);
```

属性及方法

- prototype
- isArray

原型对象属性及其方法

- constructor
- splice
- forEach
- find
- concat
- pop

- push
- reverse
- shift
- slice
- ...

Array.prototype.splice

功能：从数组中删除或添加元素，返回被删除的元素列表（作用域原有数组）

```
// arrayObject.splice(start, deleteCount[, item1[, item2[, ...]]
])
var arr = ['1', '2', 'a', 'b', '6'];
var ret = arr.splice(2, 2, '3', '4', '5'); // ['a', 'b']
arr; // ['1', '2', '3', '4', 5', '6']
```

Array.prototype.forEach

功能：遍历元素组并调用回调函数

```
// arrayObject.forEach(callback[, thisArg])
// 回调函数
// function callback(value, index, arrayObject) {...}
// value - 当前值 index - 当前索引 arrayObject - 数组本身
function logArray(value, index, arrayObject) {
  console.log(value);
  console.log(value === array[index]);
}
[2, 5, 6, 9].forEach(logArray);
```

Function

构造器说明：定义函数或新增对象构造器

实例化方法

```
// 对象实例化
var f0 = new Function('i', 'j', 'return (i + j)');
// 函数关键字语句
function f1(i, j){return i + j;}
// 函数表达式
var f3 = function(i, j){return i + j;};
```

属性及方法

- prototype

原型对象属性及其方法

- constructor
- apply
- call
- bind

实例对象属性和方法

- length
- prototype
- arguments
- caller

自定义对象构造器

下面的代码声明一个 Point 增加了一个move方法，最后创建了一个 Point 的实例对象。

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}

Point.prototype.move = function(x, y) {
  this.x += x;
  this.y += y;
}

var p = new Point(1, 2);
```

Function.prototype.apply

功能：通过参数指定调用者和函数参数并执行该函数

```
// functionObj.apply(thisArg[, argsArray])
function Point(x, y) {
  this.x = x;
  this.y = y;
}

Point.prototype.move = function(x, y) {
  this.x += x;
  this.y += y;
}

var p = new Point(1, 1);
var circle = {x: 1, y: 1, r: 1};
p.move.apply(circle, [2, 1]); // {x: 3, y: 2, r: 1}
```

Function.prototype.bind

功能：通过参数指定函数调用者和函数参数并返回该函数的引用

```
// functionObj.bind(thisArg[, arg1[, arg2[, ...]]])
function Point(x, y) {
  this.x = x;
  this.y = y;
}

Point.prototype.move = function(x, y) {
  this.x += x;
  this.y += y;
}

var p = new Point(1, 1);
var circle = {x: 1, y: 1, r: 1};
var circleMoveRef = p.move.bind(circle, 2, 1);
setTimeout(circleMoveRef, 1000); // {x: 3, y: 2, r: 1}

// 之间使用 circleMoveRef() 效果等同于 apply()
circleMoveRef();
```

子类构造器

```
function Circle(x, y, r) {
  Point.apply(this, [x, y]);
  this.radius = r;
}
Circle.prototype = Object.create(Point.prototype);
Circle.prototype.constructor = Circle;
Circle.prototype.area = function(){
  return Math.PI * this.radius * this.radius;
}

var c = new Circle(1, 2, 3);
c.move(2, 2);
c.area();
```

函数调用

- `()`

- `apply`
- `call`

函数参数

- 形参个数不一定等于实参个数
- 值专递
- 通过参数类型检查实现函数重载

arguments

arguments 的常用属性

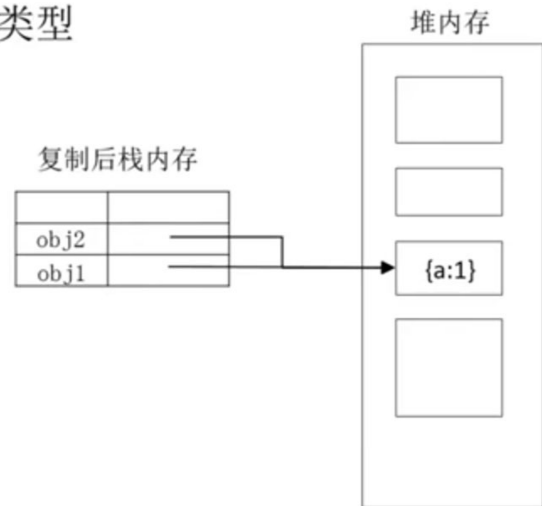
- length 实参个数
- 0...arguments.length-1 实参属性名称（key）
- callee 函数本身

```
function max(a, b) {
  if (max.length === arguments.length) {
    return a>b?a:b;
  } else {
    var _max = arguments[0];
    for(var i = 0; i < arguments.length; i++) {
      if (_max < arguments[i]) {
        _max = arguments[i];
      }
    }
    return _max;
  }
}
```

值专递

函数参数的值专递是参数复制都是栈内存中的复制。






```
// 原始类型
function plusplus(num) {
  return num++;
}
var count = 0;
var result = plusplus(count); // result = 1; count = 0;

// 引用类型
function setName(obj) {
  obj.name = 'obama';
}
var president = {name: 'bush'};
setName(president); // {name: 'obama'};
```

函数重载

以 `Require.JS` 中的 `define()` 为例：

```
define(function(){
  var add = function(x, y) {
    return x + y;
  };
  return {
    add: add
  };
})
```

```
define(['lib'], function(){
  var add = function(x, y) {
    return x + y;
  };
  return {
    add: add
  };
})

define('math', ['lib'], function(){
  var add = function(x, y) {
    return x + y;
  };
  return {
    add: add
  };
})

// define 的实现代码
/**
 * The function that handles definitions of modules. Differs fro
m
 * require() in that a string for the module should be the first
 argument,
 * and the function to execute after dependencies are loaded sho
uld
 * return a value to define the module corresponding to the firs
t argument's
 * name.
 */
define = function (name, deps, callback) {
    var node, context;

    //Allow for anonymous modules
    if (typeof name !== 'string') {
        //Adjust args appropriately
        callback = deps;
        deps = name;
        name = null;
    }
```

```
    //This module may not have dependencies
    if (!isArray(deps)) {
        callback = deps;
        deps = null;
    }

    // 省略以下代码
    // ...
};
```

RegExp

构造器说明：用于定义正则表达式，一个 RegExp 对象包含一个正则表达式和关联的标志

定义方法

- `/pattern/flags`
- `new RegExp(pattern[, flags]);`

属性及方法

- prototype

原型对象属性及其方法

- constructor
- test
- exec
- ...

RegExp.prototype.test

功能：使用正则表达式对字符串进行测试，并返回测试结果

```
// regexObj.text(str)
var reg = /^abc/i;
reg.test('Abc123'); // true
reg.test('1Abc1234'); // false
```

Date

构造器说明：用于定义日期对象

定义方法

```
var date0 = new Date();
var date1 = new Date(2014, 3, 1, 7, 1, 1, 100);
```

属性及方法

- prototype
- parse
- now
- ...

原型对象属性及其方法

- constructor
- Date
- getDate
- getHours
- setDate
- setHours
- ...

标准内置对象

Math

对象说明：拥有属性和方法的单一对象主要用于数字计算

对象属性：

- E
- PI
- SQRT2
- ...

对象方法：

- floor
- random
- abs
- max
- cos
- ceil

Math.floor

功能：向下取整

```
// Math.floor(num)
Math.floor(0.97); // 0
Math.floor(5.1); // 5
Math.floor(-5.1); //6
```

相似方法：`ceil`，`round`

Math.random

功能：返回 0~1 之间的浮点数

```
// Math.random()
Math.random(); // 0.14523562323461
```

JSON

对象说明：用于存储和交换文本信息

对象方法：

- parse
- stringify

JSON.stringify

功能：将 JSON 对象转换为字符转

```
// JSON.stringify(value[, replacer[, space]])
var json = {'name': 'X'};
JSON.stringify(json); // "{"name":"X"}"
```

JSON.parse

功能：将 JSON 字符转转换为对象

```
// JSON.parse(text[, reviver])
var jsonStr = '{"name":"X"}';
JSON.parse(jsonStr); // {name: 'X'}
```

全局对象

全局对象定义了一系列的属性和方法在编程过程中可以被之间调用。

属性：NaN，Infinity，undefined

方法：

- parseInt
- parseFloat
- isNaN
- isFinite
- eval

处理 URI 方法：

- encodedURIComponent
- decodeURIComponent
- encodedURI
- decodeURI

构造器属性：

- Boolean
- String
- Number
- Object
- Function
- Array
- Date
- Error
- ...

对象属性：

- Math
- JSON

NaA

非数字值：表示错误或无意义的运算结果，NaN 参与运算仍会返回 NaA，且 NaN 不等于任何值，包括它本身。可以使用 `isNaN()` 判断运算结果的类型是否为 NaN。

```
isNaN(NaN); // true
isNaN(4 - '2a'); // true;
```

parseInt

功能：转换字符串成数字

```
// parseInt(string[, radix])
// radix - 为进制数
parseInt('010'); // 10
parseInt('010', 8) // 8
parseInt('010', 16) // 16

parseInt('0x1f'); // 31
parseInt('0x1f', 16); // 31
parseInt('1f'); // 1
parseInt('1f', 16); // 31
```

eval

功能：计算字符串并执行其中的 JavaScript 代码（会带来安全性和代码逻辑问题，通常不建议使用）

```
// eval(string)
var res = '{"error": "0", "msg": "OK"};
var obj;
if (!JSON) {
  obj = eval('(' + res + ')');
} else {
  obj = JSON.parse(res);
}
```

encodedURIComponent

功能：将 URI 参数中的特殊字符，中文等作为 URI 的一部分进行编码

```
var uri = "http://w3schools.com/my test.asp?name=ståle&car=saab"
;
var res = encodeURIComponent(uri);

// 结果
// http%3A%2F%2Fw3schools.com%2Fmy%20test.asp%3Fname%3Dst%C3%A5l
e%26car%3Dsaab
```

Table of Contents *generated with DocToc*

变量作用域

变量的作用域值的是变量的生命周期和作用范围（全局与局部作用域的区别）。

作用域介绍

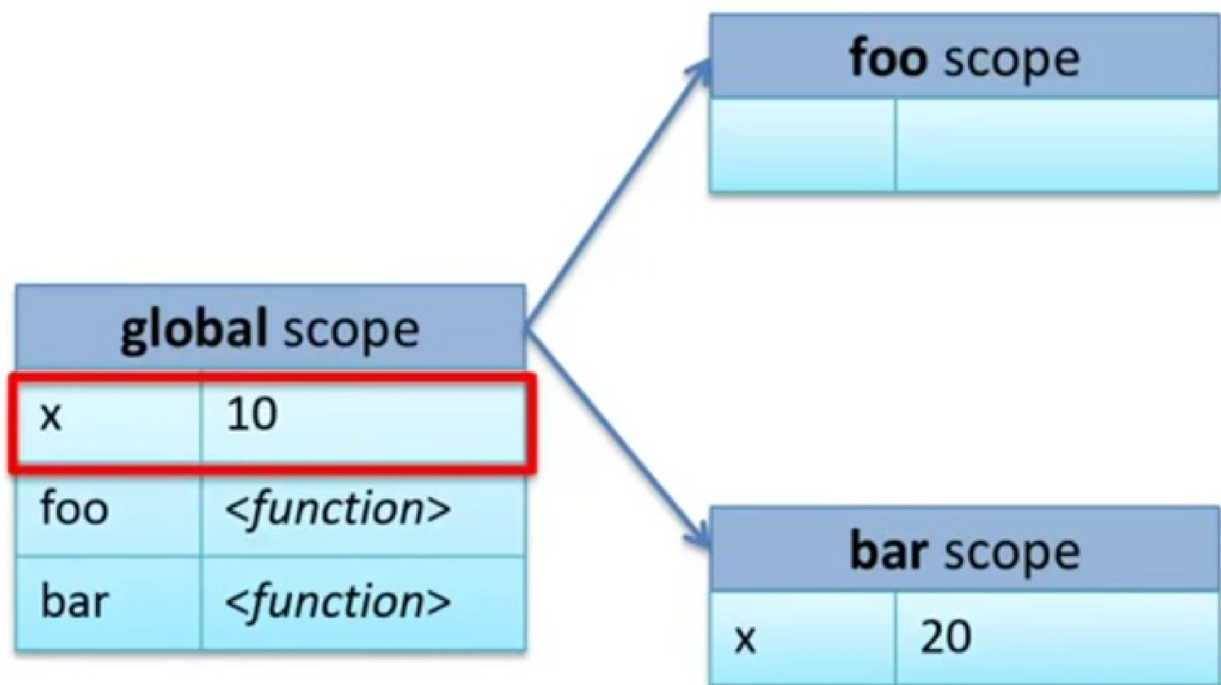
静态作用域

静态作用域有称为词法作用域，即指其在编译的阶段就可以决定变量的引用。静态作用域只更变量定义的位置有关与代码执行的顺序无关。

```
var x = 0;
function foo() {
  alert(x);
}

function bar() {
  var x = 20;
  foo();
}

foo();
```



动态作用域

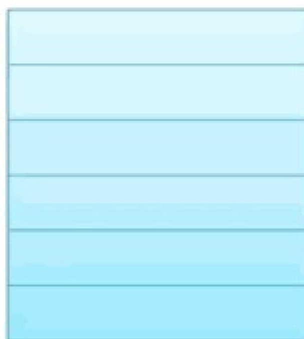
动态作用域的变量引用只可在程序运行时刻决定（其通常通过动态栈来进行管理）。

```
var x = 0;
function foo() {
  alert(x);
}

function bar() {
  var x = 20;
  foo();
}

foo();
```

```
var x=10;
function foo(){
    alert(x);
}
function bar(){
    var x=20;
    foo();
}
bar();
```



JavaScript 变量作用域

JavaScript（1）使用静态作用域，（2）其没有块级作用域（只有函数作用域，就是只有 function 用于可以定义作用域），（3）在 ES5 之作使用词法环境来管理作用域。

词法环境

组成

词法环境用来描述静态作用域的数据结构。它由 `环节记录` 和 `外部词法环境的引用` 组成。

- 环境记录（record）（指形参，变量，函数等）
- 外部词法环境的引用（outer）

创建

在一段代码执行之前，先初始化词法环境。会被初始化的有：

- 形参
- 函数定义（创建函数对象，会保存当前作用域。见下图）
- 变量定义（所有初始化值均为 `undefined` ）

<function>	
formalParameter	...
functionBody	...
scope	Current Environment

结构

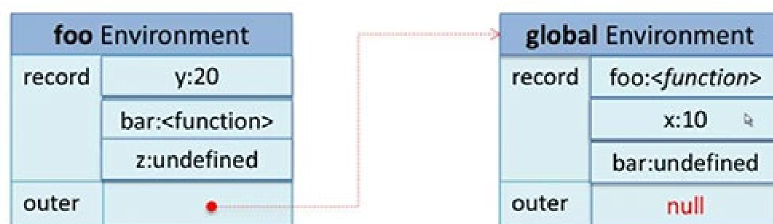
```
var x = 10;
function foo(y) {
  var z = 30;
  function bar(q) {
    return x + y + z + q;
  }
  return bar;
}
var bar = foo(20);
bar(40);
```

全局词法作用域（初始化状态）

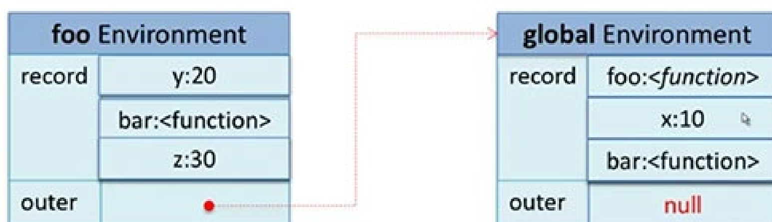
global Environment	
record	foo:<function>
	x:undefined
	bar:undefined
outer	null

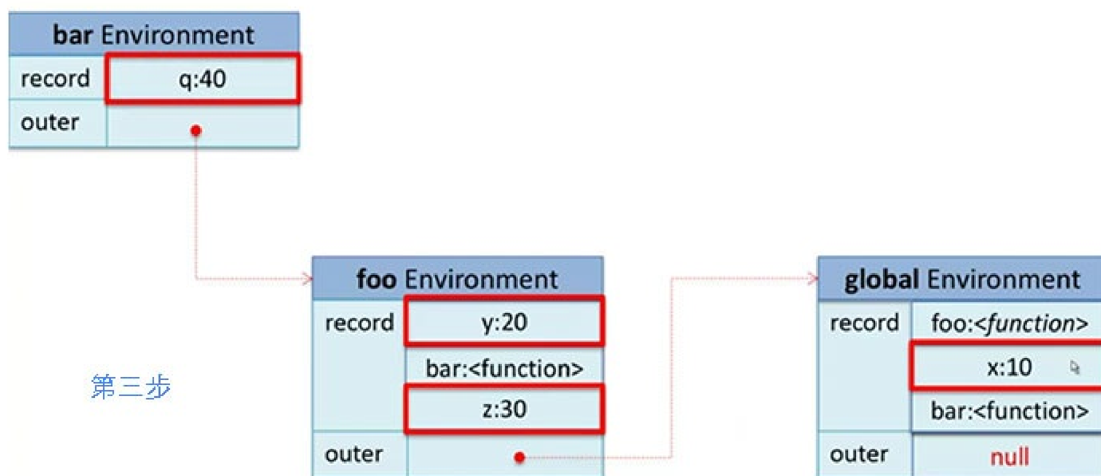
函数词法作用域

第一步



第二步





关于词法环境的问题

命名冲突

形参，函数定义，变量名称命名冲突。其中的优先级的排序如下：

```
函数定义 > 形参 > 变量
```

`arguments` 的使用

为函数中定义好的变量。

函数表达式与函数定义的区别

- 函数表达式是在执行时才创建函数对象。
- 函数定义为在代码执行之前就进行创建的。

with 语句

`with` 会创造一个临时作用域。

```
var foo = 'abc';
with({
  foo: 'bar';
}) {
  function f() {
    alert(foo);
  };
  (function() {
    alert(foo);
  })();
  f();
}
```

try-catch 句法

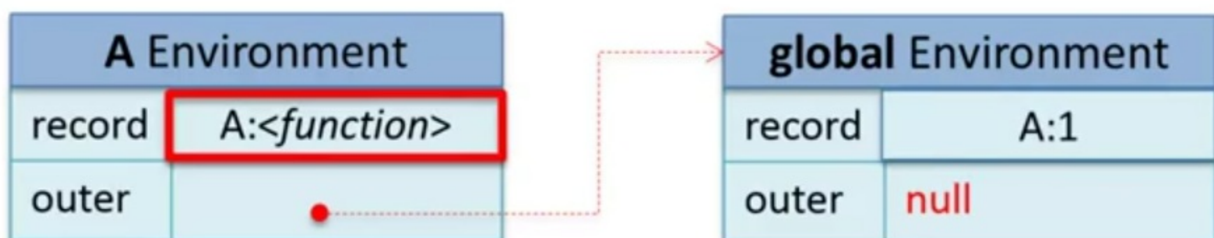
```
try {
  var e = 10;
  throw new Error();
} catch (e) {
  function f() {
    alert(e);
  }
  (function() {
    alert(e);
  })();
  f();
}
```

带名称的函数表达式

当一个函数表达式有了名称之后，JavaScript 会创建一个新的词法环境。并在这个词法环境中用有一个属性 A 指向这个函数，同时这个属性 A 指向的函数是不可被修改的。

下面例子为不常规的写法

```
(function A(){
  A = 1;
  alert(A);
})();
```



表达式与运算符

表达式

表达式为 JavaScript 的短语可执行并生成值。

```
1.7 // 字面量
"1.7"
var a = 1;
var b = '2';
var c = (1.7 + a) * '3' - b
```

运算符

- 算数运算符 （`+` `-` `*` `/` `%`）
- 关系运算符 （`>` `<` `==` `!=` `>=` `<=` `===` `!==`）
- 逻辑运算符 （`!` `&&` `||`）
- 位运算符 （`&` `|` `^` `~` `<<` `>>`）
- 负值运算符 （`=`）
- 条件运算符 （`?:`）
- 逗号运算符 （`,`）
- 对象运算符 （`new` `delete` `.` `[]` `instanceof`）

=== 全等符号

全等运算符用于盘对左右两边的对象或值是否类型相同且值相等。

伪代码拆解

```
function totalEqual(a, b) {
  if (a 和 b 类型相同) {
    if (a 和 b 是引用类型) {
      if (a 和 b 是同一引用)
        return true;
      else
        return false;
    } else { // 值类型
      if (a 和 b 值相等)
        return true;
      else
        return false;
    }
  } else {
    return false;
  }
}
```

例子

```
var a = "123";
var b = "123";
var c = "4";
var aObj = new String("123");
var bObj = new String("123");
var cObj = aObj;

a === aObj      // false
aObj === bObj   // false
aObj === cObj   // true
a === b         // true
a === c         // false
```

==

`==` 用于判断操作符两边的对象或值是否相等。

伪代码拆解

```
function equal(a, b) {
  if (a 和 b 类型相同) {
    return a === b;
  } else { // 类型不同
    return Number(a) === Number(b); // 优先转换数值类型
  }
}
```

例子

```
"99" == 99; // true
new String("99") == 99; // true
true == 1; // true
false == 0; // true
'\n\n\n' == // true
```

例外规则

- `null == undefined` 结果为真 `true`
- 在有 `null` / `undefined` 参与的 `==` 运算是不进行隐式转换。

```
0 == null; // false
null == false; // false
"undefined" == undefined; // false
```

! 取反

`!x` 用于表达 x 表达式的运行结果转换成布尔值（Boolean）之后取反的结果。 `!!x` 则表示取 x 表达式的运行结果的布尔值。

```
var obj = {};
var a = !obj // false;
var a = !!obj // true;
```

&& 逻辑与

`x && y` 如果 x 表达式的运行交过转换成 Boolean 值为 false 则不运行表达式 y 而直接返回 x 表达式的运行结果。相反，如果 x 表达式的运行交过转换成 Boolean 值为 true 则运行表达式 y 并返回 y 表达式的运行结果。

伪代码拆解

```
var ret = null;
if (!!(ret = x)) {
  return y;
} else {
  return ret;
}
```

例子

```
var a = 0 && (function(){return 1 + 1;})(); // 0
var b = 1 && (function(){return 1 + 1;})(); // 2
```

|| 逻辑或

`x || y` 如果 x 表达式的运行结果转换为 Boolean 值为 true，则不运行 表达式 y 而直接返回表达式 x 的运算结果。（与 `&&` 方式相反）

伪代码拆解

```
var ret = null;
if (!!(ret = x)) {
  return ret;
} else {
  return y;
}
```

例子

```
var a = 0 || (function(){return 1 + 1;})(); // 2
var b = 1 || (function(){return 1 + 1;})(); // 1
```

元算符优先级（Operator Precedence）

- `+` `-` `*` `/` 高于 `&&`
- `*` `/` 高于 `+` `-`
- `&&` 高于 `?:`
- `()` 内优先级高于之外

NOTE：和数学上的算术优先级类似，同级从左到右计算。如有疑问加上 `()` 既可解决优先级问题。

Precedence	Operator type	Associativity	Individual operators
19	Grouping	n/a	`( … )`
18	Member Access	left-to-right	`… . …`
	Computed Member Access	left-to-right	`… [ … ]`
	new (with argument list)	n/a	`new … ( … )`
17	Function Call	left-to-right	`… ( … )`
	new (without argument list)	right-to-left	`new …`
16	Postfix Increment	n/a	`… ++`
	Postfix Decrement	n/a	`… --`
15	Logical NOT	right-to-left	`! …`
	Bitwise NOT	right-to-left	`~ …`
	Unary Plus	right-to-left	`+ …`
	Unary Negation	right-to-left	`- …`
	Prefix Increment	right-to-left	`++ …`
	Prefix Decrement	right-to-left	`-- …`
	typeof	right-to-left	`typeof …`
	void	right-to-left	`void …`
	delete	right-to-left	`delete …`
14	Multiplication	left-to-right	`… * …`
	Division	left-to-right	`… / …`

	Remainder	left-to-right	`… % …`
13	Addition	left-to-right	`… + …`
	Subtraction	left-to-right	`… - …`
12	Bitwise Left Shift	left-to-right	`… << …`
	Bitwise Right Shift	left-to-right	`… >> …`
	Bitwise Unsigned Right Shift	left-to-right	`… >>> …`
11	Less Than	left-to-right	`… < …`
	Less Than Or Equal	left-to-right	`… <= …`
	Greater Than	left-to-right	`… > …`
	Greater Than Or Equal	left-to-right	`… >= …`
	in	left-to-right	`… in …`
	instanceof	left-to-right	`… instanceof …`
10	Equality	left-to-right	`… == …`
	Inequality	left-to-right	`… != …`
	Strict Equality	left-to-right	`… === …`
	Strict Inequality	left-to-right	`… !== …`
9	Bitwise AND	left-to-right	`… & …`
8	Bitwise XOR	left-to-right	`… ^ …`
7	Bitwise OR	left-to-right	`… | …`
6	Logical AND	left-to-right	`… && …`
5	Logical OR	left-to-right	`… || …`
4	Conditional	right-to-left	`… ? … : …`
3	Assignment	right-to-left	`… = …`
			`… += …`
			`… -= …`
			`… *= …`
			`… /= …`
			`… %= …`
			`… <<= …`

<table>
<tr><td></td><td></td><td></td><td>… >>= …</td></tr>
<tr><td></td><td></td><td></td><td>… >>>= …</td></tr>
<tr><td></td><td></td><td></td><td>… &= …</td></tr>
<tr><td></td><td></td><td></td><td>… ^= …</td></tr>
<tr><td></td><td></td><td></td><td>… |= …</td></tr>
<tr><td>2</td><td>yield</td><td>right-to-left</td><td>yield …</td></tr>
<tr><td>1</td><td>Spread</td><td>n/a</td><td>... …</td></tr>
<tr><td>0</td><td>Comma / Sequence</td><td>left-to-right</td><td>… , …</td></tr>
</table>

语句

条件控制语句

其中expression可以使用整型，字符串，甚至表达式

```
if (expression0) {statement0}
else if (expression1) {statement1}
else {statement2}

// JavaScript 中的 case 可以使用整型，字符串，甚至表达式
switch(persion.type) {
  case "teacher":
    statement1
    break;
  case "student":
    statement2
    break;
  default:
    statement3
    break;
}
```

循环控制语句

```
while(expression) {statement}

// 至少执行一次
do {statement} while(expression);

for (initialise; test_expresiion; increment) {statement}

// 跳过下面代码并进入下一轮循环
continue;

// 退出当前循环
break;
```

for-in

用于遍历对象的全部属性。

```
function Car(id, type, color) {
  this.type = type;
  this.color = color;
  this.id = id;
}

var benz = new Car('benz', 'black', 'red');
Car.prototype.start = function(){
    console.log(this.type + ' start');
}

for (var key in benz) {
  console.log(key + ':' benz[key]);
}

// 输出结果
type:black
color:red
id:benz
start:function (){
    console.log(this.type + ' start');
}

// -----------

// 如不需原型对象上的属性可以使用 hasOwnProperty
for (var key in benz) {
  if (benz.hasOwnProperty(key)) {
    console.log(key + ':' benz[key]);
  }
}

// 输出结果
type:black
color:red
id:benz
```

如不需原型对象上的属性可以使用 hasOwnProperty

```
for (var key in benz) {
  if (benz.hasOwnProperty(key)) {
    console.log(key + ':' benz[key]);
  }
}
/* 输出结果
type:black
color:red
id:benz */
```

异常处理语句

```
try{
  // statements
  throw new Error();
catch(e){
  // statements
}
finally{
  // statements
}
```

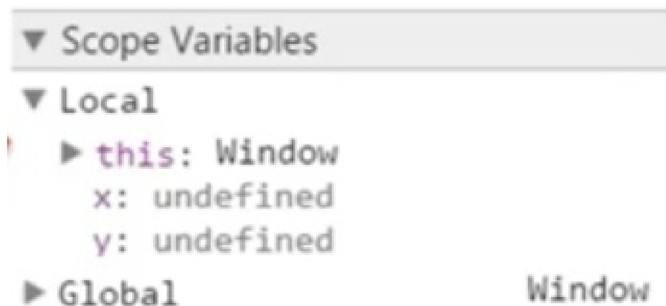
with 语句

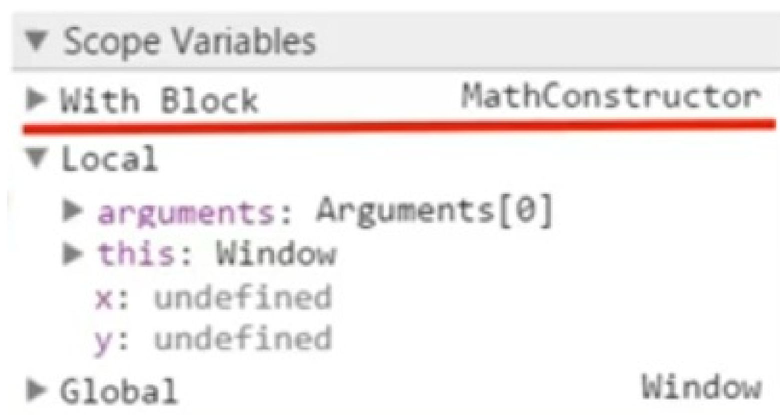
`with` 语句是 JavaScript 中特有的语句形式，它主要有两个作用：

其一，其用于缩短特定情况下必须书写的代码量。它可以暂时改变变量的作用域。

```
// 使用 with 之前
(function(){
  var x = Math.cos(3 * Math.PI) + Math.sin(Math.LN10);
  var y = Math.tan(14 * Math.E);
})();

// 使用 with
(function(){
  with(Math) {
    var x = cos(3 * PI) + sin(LN10);
    var y = tan(14 * E);
  }
})();
```





其二，改变变量的作用域，将 `with` 语句中的对象添加至作用域链的头部。

```
frame[1].document.forms[0].name.value = "";
frame[1].document.forms[0].address.value = "";
frame[1].document.forms[0].email.value = "";

with(frame[1].document.[0]) {
  name.value = "";
  address.value = ""
  email.value = "";
}
```

缺点就是导致 JavaScript 语句的可执行性下降，所以通常情况下因尽可能的避免使用。

Table of Contents *generated with DocToc*

闭包

- 闭包有函数和与其相关的引用环境的组合而成
- 闭包允许函数访问其引用环境中的变量（又称自由变量）
- 广义上来说，所有 JavaScript 的函数都可以成为闭包，因为 JavaScript 函数在创建时保存了当前的词法环境。

```
function add() {
  var i = 0;
  return function() {
    alert(i++);
  }
}
var f = add();
f();
f();
```

闭包的应用

保存变量现场

```
// 错误方法
var addHandlers = function(nodes) {
  for (var i = 0, len = nodes.length; i < len; i++) {
    nodes[i].onclick = function(){
      alert(i);
    }
  }
}

// 正确方法
var addHandlers = function(nodes) {
  var helper = function(i) {
    return function() {
      alert(i);
    }
  }

  var (var i = 0, len = nodes.length; i < len; i++) {
    nodes[i].onclick = helper(i);
  }
}
```

封装

```
// 将 observerList 封装在 observer 中
var observer = (function(){
  var observerList = [];
  return {
    add: function(obj) {
      observerList.push(obj);
    },
    empty: function() {
      observerList = [];
    },
    getCount: function() {
      return observerList.length;
    },
    get: function() {
      return observerList;
    }
  };
})();
```

Table of Contents *generated with DocToc*

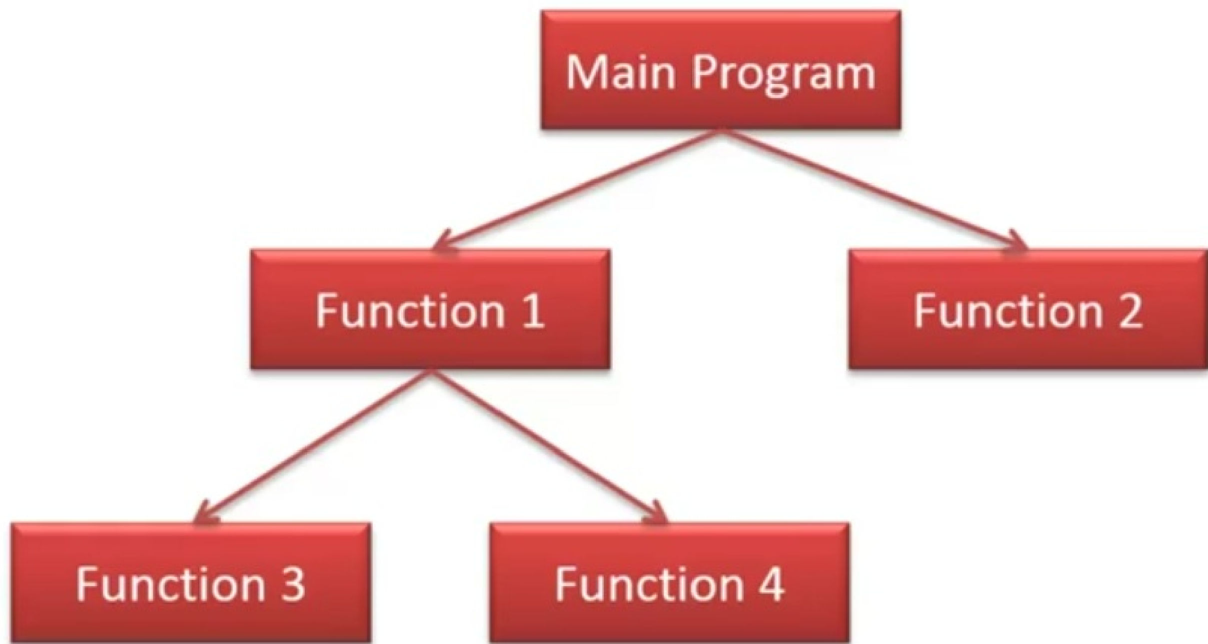
面向对象

程序设计方法

程序设计描述系统如何通过程序来实现的过程，其为一种设计方法与语言实现无关。常见的设计方法有面向流程与面向对象。

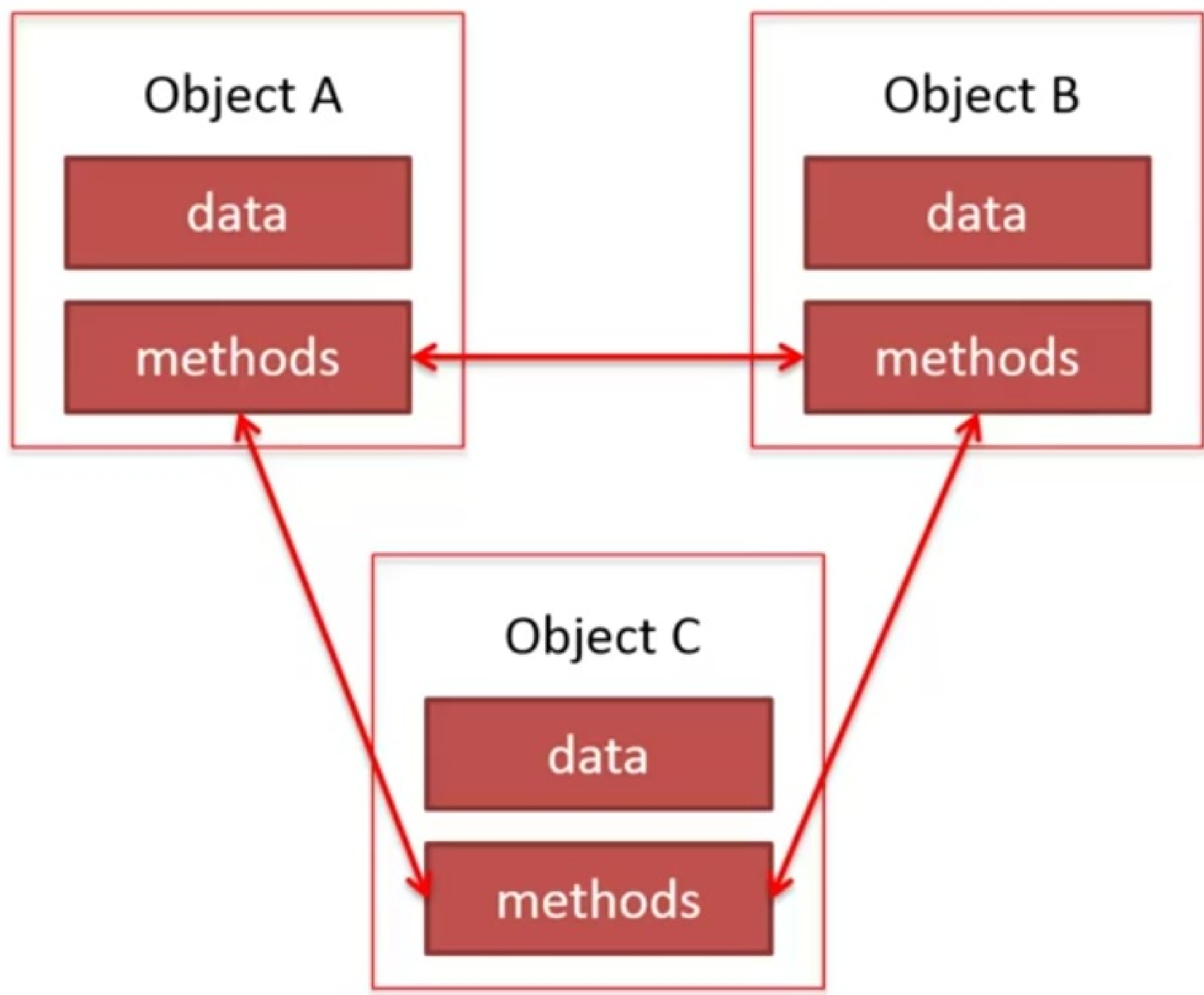
面向过程

以程序的过程为中心，采用自定而下逐步细化的方法来实现。常见的面向过程语言有 C、Fortran、Pascall。



面向对象

将对象作为程序的基本单元，将程序分解为数据和操作的集合。常见的面向过程语言有 smalltalk（也是 Objective-C 的父亲）、Java、C++。



概念

- 类（Class）、对象（Object）
- 属性（Property）、方法（Method）

基本特点

- 继承（Inheritance）
- 封装（Encapsulation）
- 多态（Polymorphism）

JavaScript 面向对象

constructor

对象的构造器，也可称之为构造类型。

```
// 使用 new 关键字创建
var o = new Object();
var a = new Array();
var d = new Date();
    |          |
 object     constructor

// 使用直接量创建
var o = {name: 'Xinyang'};
var a = [1, 2, 3];
```

自定义构造器

```
// constructor
function Person(name, age, birthdate) {
  this.name = name;
  this.age = age;
  this.birthdate = birthdate;
  this.changeName = function(newAge) {
    this.age = newAge;
  }
}

// 创建对象
var X = new Person('Stupid', 13, new Date(2015, 01, 01));
var Q = new Person('Q', 12, new Date(2015, 01, 01));

X.changeName('X');
```

创建构造器的方法（**3** 种）

- `function ClassName() {...}`
- `var Class = function() {...}`
- `var Class = new Function()`

NOTE: 并不是所有函数都可以被当成构造器，例如 `var o = new Math.min()` 。通常自定义的函数均可当做构造器来使用。内置对象的构造器也可被当做构造器。

NOTE+：如果构造器有返还值并为对象类型，则对象将被直接返回。

```
function Person(name, age, birthdate) {
  this.name = name;
  this.age = age;
  this.birthdate = birthdate;
  this.changeName = function(newAge) {
    this.age = newAge;
  }
  // !!! 注意这里
  return {};
}

var X = new Person('X', 13, new Date());
console.log(X.name); // undefined;
```

this

`this` 在不同环境中指代的对象不同（ `this` 指代的值可在函数运行过程中发生改变）。

出现场景	所指代值
全局环境	全局对象（window 在浏览器环境中时）
constructor	创建的新实例对象
函数调用	函数的调用者
`new Function()`	全局对象
`eval()`	调用上下文中的 `this`

全局环境中

全局环境中 `this` 指代全局对象，既 `window` 在浏览器环境中。

```
// 以下的所有 this 均指代全局对象
var a = 10;
alert(this.a);

this.b = 20;
alert(b);

c = 30;
alert(this.c);
```

构造器中

构造器中的 `this` 指代的是即将被创建出的对象。

```
// constructor
function Person(name, age, birthdate) {
  // 下面的指代即将被创建的对象
  this.name = name;
  this.age = age;
  this.birthdate = birthdate;
  this.changeName = function(newAge) {
    this.age = newAge;
  }
}

// 创建对象
var X = new Person('Stupid', 13, new Date(2015, 01, 01));
var Q = new Person('Q', 12, new Date(2015, 01, 01));

X.changeName('X');
```

函数中

函数中的 `this` 指代函数的调用者。

```
// constructor
function Person(name, age, birthdate) {
  // 下面的指代即将被创建的对象
  this.name = name;
  this.age = age;
  this.birthdate = birthdate;
  this.changeName = function(newAge) {
    this.age = newAge;
  }
  this.gretting = function() {
    // !!! 下面这个 this 指代调用它的对象，既上面的
    // 上面的 gretting 左边的 this，既为即将被创建的对象
    console.log('Hi, I am ' + this.name)
  }
}

// 创建对象
var X = new Person('Stupid', 13, new Date(2015, 01, 01));

X.changeName('X');
X.gretting();
```

NOTE: `new Function('console.log(this)')` 中的 `this` 均指代全局对象。`eval('console.log(this)` 则为调用上下文指代的 `this`。

this 实例

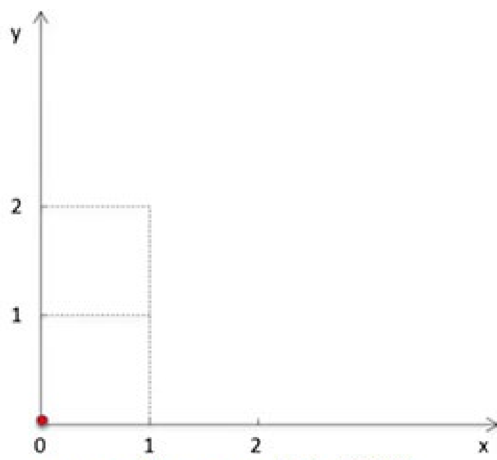
下面的例子使用 `apply` 与 `call` 。通过这两个方法来将一个对象中 `this` 指代的目标进行改变。

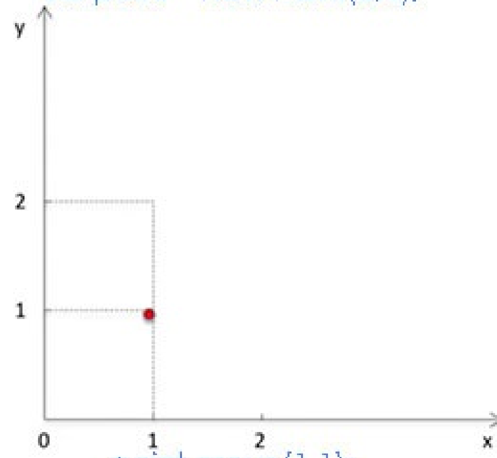
```
function Point(x, y) {
  this.x = x;
  this.y = y;
  this.move = function(x, y) {
    this.x += x;
    this.y += y;
  }
}

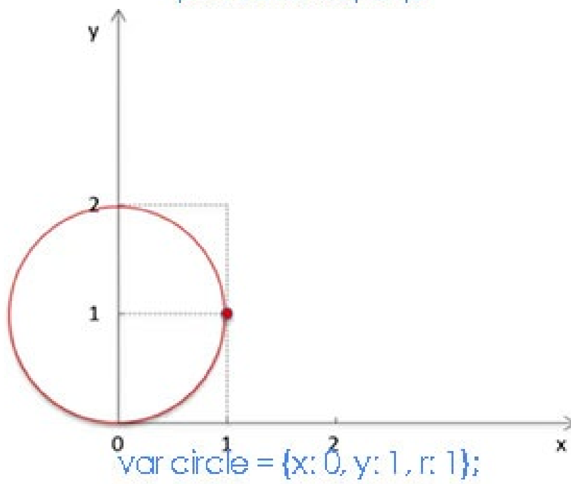
var point = new Point(0, 0);
point.move(1, 1);

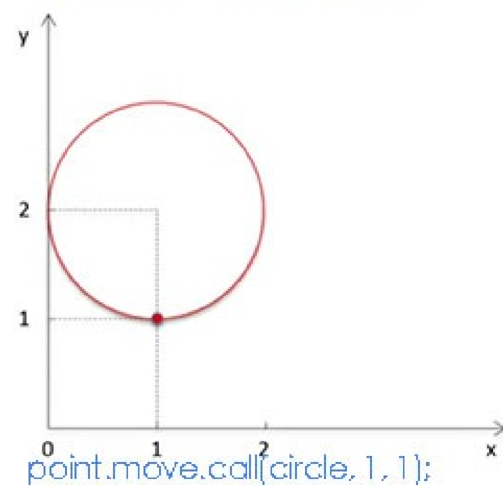
var circle = {x: 0, y: 1, r: 1};

// 改变 point 中 move 方法 this 指代的对象至 circle
point.move.apply(circle, [1, 1]);
// 同样可以用类似的 call 方法，区别为参数需依次传入
point.move.call(circle, 1, 1);
```


y
2
1
0
1
2
x
var point = new Point(0,0);


y
2
1
0
1
2
x
point.move(1,1);


y
2
1
0
1
2
x
var circle = {x: 0, y: 1, r: 1};


y
2
1
0
1
2
x
point.move.call(circle, 1, 1);

原型继承

使用原型（prototype）可以解决重复定义实例对象拥有的完全一致的属性或方法（既共享原型中的属性或方法）。

```
function Boss() {
  this.age = 0;
  this.birthdate = null;
  this.name = '';
  this.tasks = [];
  this.title = 'Boss';
  this.gretting = function() {console.log('I am a Boss!');};
}

var X = new Boss();
var Q = new Boss();

// X 与 Q 中具有完全一致（不必唯一的属性或方法）
// 并耗用内存的共享部分
// this.title 与 this.gretting
```

改造后的构造器

```
function Boss() {
  this.age = 0;
  this.birthdate = null;
  this.name = '';
  this.tasks = [];
}
Boss.prototype = {
  title: 'Boss',
  gretting: function(){console.log('I am a Boss!');}
}

var X = new Boss();
var Q = new Boss();

// X 与 Q 中具有完全一致（不必唯一的属性或方法）
// 并耗用内存的共享部分
// this.title 与 this.gretting

var X = new Boss();
var Q = new Boss();

// X 与 Q 拥有相同的原型 Boss.prototype
```

原型链

使用原型继承的方法会产生原型链。JavaScript 中对于对象的查找、修改和删除都是通过原型链来完成的。

判断属性是否为对象本身

```
objectName.hasOwnProperty('propertyName');
// 返回布尔值 true 或 false
```

属性查找

对象的属性查找会更随原型链依次查找，如果在当前环境中无法找到需要的属性则会继续向下一层原型中继续寻找。

属性修改

在 JavaScript 中对于对象属性的修改永远只修改对象自身的属性（不论是来源于对象本身还是对象的原型）。当创建当前对象不存在属性时（即便原型拥有此属性），也会为此对象增加改属性。

修改原型上的属性

修改原型属性会印象所有被创建出的对象现有的属性和方法。

```
ClassName.prototype.propertyName = 'new value';
ClassName.prototype.methodName = function(){...};
```

属性删除

`delete objectName.propertyName` 只可删除对象自身的属性，无法删除对象的原型属性。

Object.create(proto[, propertiesObject])

其为ECMAScript 5 中提出的新建立对象的方式。在 `X` 中使用隐式的原型对象指向 `boss` 对象，并将其设为 `X` 对象的原型。

```
var boss = {
  title: 'Boss',
  gretting: function(){console.log('Hi, I am a Boss!');}
};

var X = Object.create(boss);
X.gretting(); // Hi, I am a Boss!
```

低版本中实现 **Object.create** 功能

此种方式仍需使用 `ClassName.prototype` 的方式来实现。

```
var clone = (function(){
  var F = function(){};
  return function(proto) {
    F.prototype = proto;
    return new F();
  }
})();
```

面向对象的应用

全局变量

全局变量可在程序任意位置进行访问和修改的变量。滥用全局变量会导致，命名冲突，导致程序不稳定。

全局标量的三种定义方法：

- `var gloablVal = 'value';` 。
- `window.gloablVal = 'value';` 附加于 `window` 对象上
- `gloablVal = 'value';` 不使用 `var` 关键字，也附加于 `windwo` 对象

NOTE： `delete` 无法删除在代码最顶端定义的全局标量 `var globale`

封装

信息隐藏可以保证程序的稳定，将内部信息进行隐藏。其他语言中可词用访问权限来实现封装的概念，像 `private` 、 `public` 。

JavaScript 中的封装可使用函数的方法（闭包）。

```
// 模拟 private 的属性
function ClassName(){
  var _property = '';
  this.getProperty = function(){
    return _property;
  };
}

// 模拟 protected 属性，使用人为约束规则
var pro = ClassName.prototype;
pro._protectedMethod = function(){...};
pro.publicMethod = function(){...};
```

继承

原型继承

原型继承的方式为 JavaScript 中固有的继承方式。

```
var proto = {
  action1: function(){},
  action2: function(){}
}

var obj = Object.create(proto);
```

在不支持 EM5 中的实现方法：

```
var clone = (function(){
  var F = function(){};
  return function(proto) {
    F.prototype = proto;
    return new F();
  }
})();
```

类继承

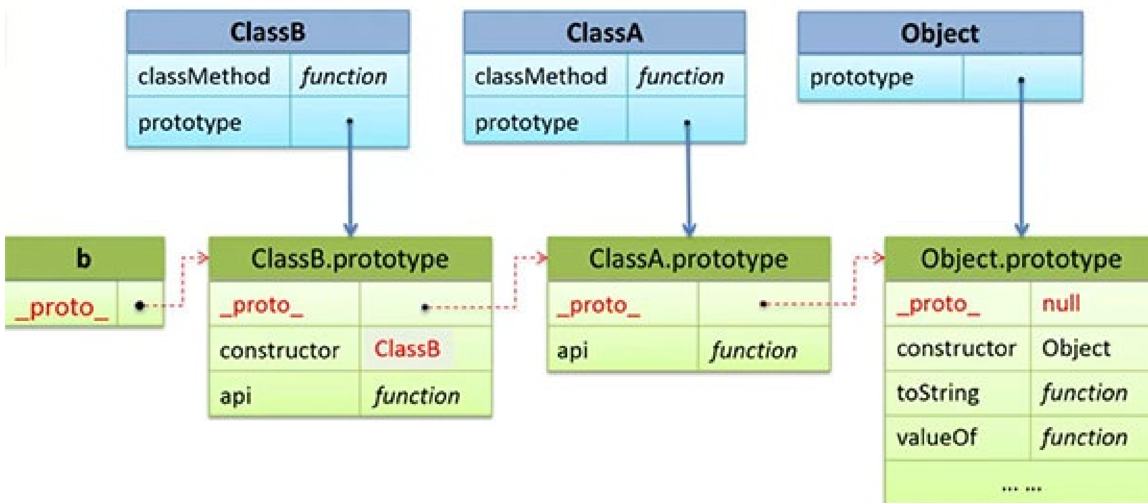
使用原型链继承的方式模拟其他语言类继承的特性。

```
function ClassA() {
  ClassA.classMethod = function(){};
  ClassA.prototype.api = function(){};

  function ClassB() {
    ClassA.apply(this, argument);
  }
  ClassB.prototype = new ClassA();
  ClassB.prototype.constructor = ClassB;
  ClassB.prototype.api = function(){
    ClassA.prototype.api.apply(this, arguments);
  }
}

// ClassA 为父类
// ClassB 为子类

var b = new ClassB();
b.api();
```



正则表达式

正则表达式为标书字符串规则的表达式。下面为两个例子为在 JavaScript 中的使用，

- `/pattern/attrs`
- `new RegExp(pattern, attrs)`

锚点

锚点用于匹配一个位置，下列为常用的锚点

- `^` 起始的位置 `/^http/`
- `$` 结尾的位置 `/\.jpg$/`
- `\b` 单词边界

字符类

字符类用于匹配一类字符中的一个，下面为几个常用的例子，

- `[abc]` 可用于匹配 `a` `b` 还有 `c`
- `[0-9]` 可用于匹配一个数字
- `[^0-9]` 可用于匹配一个非数字
- `[a-z]` 可用于匹配一个字母
- `.` 任意字符但换行符除外

元字符

元字符为具有特殊意义的字符。常见的有，

- `^`，`$`，`\b`
- `\d` 用于匹配数字 `[0-9]`
- `\D` 用于匹配 `[^\d]`
- `\s` 用于匹配空白符
- `\S` 用于匹配非空白符 `[^\s]`
- `\w` 用于匹配任意单词字符（例如程序中的变量字符） `[A-Za-z0-9_]`

- `\W` 用于匹配非单词字符 `[^\W]`

量词

量词用于表现字符出现的次数。可用的连词如下，

- `{m,n}` 用于表示出现 `m` 到 `n` 次之间。
- `*` 用于表示出现 `0` 到无穷之间也就等同于 `{0,}`
- `?` 用于表示出现 `0` 次到 `1` 次也等同于 `{0,1}`
- `+` 用于表现出现 `1` 次以及一次以上也等同于 `{1,}`

转移符

转义符需要在匹配的字符是元字符的时候使用。使用 `\` 来进行转移即可。

多选分支

多选分支用于表示或的概念。`/thi(c|n)k/` 其又等同于 `/thi[cn]k/`。其还可以用于匹配文件扩展名 `/\.(png|jpg|jpeg|gif)$/` 。

常用方法

测试：**regxObj.test(str)**

其用于测试正则表达式与指定字符串是否匹配。

```
/123/.test(123); // true
/123/.test(111); // false
/123/.test('x123'); // true
```

捕获

其用于保存所匹配到的字符串为后续开发所用。`()` 可用于捕获，正则表达式再运行时会将其保存下来，`(?:)` 则不予保存。

```
// str.match(regexp)
var url = 'http://www.google.com/query?test=li-xinyang#cool';
var reg = /(https?:)\/\/([^\/]+)(\/[^\?]*)?(\?[^#]*)?(#.*)?/;

var arr = url.match(reg);
var protocol = arr[1];
var host = arr[2];
var pathname = arr[3];
var search = arr[4];
var hash = arr[5];
```

搜索与替换

`regexpObj.exec(str)` 可以提供更强大的检索，它可以提供更详尽的结果 `index` 也可以提供过程状态 `lastIndex` 。

`str.replace(regex/substr, replacement)` 可以使用正则表达式来对字符串进行替换。

Table of Contents *generated with DocToc*

DOM 编程艺术

DOM 编程就是使用 W3C 定义的 API (Application Program Interface) 来操作 HTML 文档 （此处不局限于 HTML，亦可操作 XHTML、XML 等），使用户可以与进行页面交互。 你需要了解节点、属性、样式等基本 DOM 操作，DOM 事件模型，数据存储 (Cookie、Storage) 与数据通信 (Ajax) ，JavaScript 动画，音频、视频、Canvas 等 HTML5 特性，表单、列表操作。

Table of Contents *generated with DocToc*

文档树

Document Object Model (DOM) 为文档对象模型， 它使用对象的表示方式来表示对应的文档结构及其中的内容。

下面为一个样例 `p` 元素在文档中的对象所包含的所有属性。

```
<p id="target">Hello, World!</p>
```

```
p#targetaccessKey: ""
align: ""
attributes: Named
NodeMapbaseURI: ""
childElementCount: 0
childNodes: NodeList[1]
children: HTMLCollection[0]
classList: DOMTokenList[0]
className: ""
clientHeight: 0
clientLeft: 0
clientTop: 0
clientWidth: 0
contentEditable: "inherit"
dataset: DOM
StringMapdir: ""
draggable: false
firstChild: text
firstElementChild: null
hidden: false
id: "target"
```

```
innerHTML: "Hello, World!"
innerText: "Hello, World!"
isContentEditable: false
lang: ""
lastChild: text
lastElementChild: null
localName: "p"
namespaceURI: "http://www.w3.org/1999/xhtml"
nextElementSibling: null
nextSibling: null
nodeName: "P"
nodeType: 1
nodeValue: null
offsetHeight: 0
offsetLeft: 0
offsetParent: null
offsetTop: 0
offsetWidth: 0
onabort: null
onautocomplete: null
onautocompleteerror: null
onbeforecopy: null
onbeforecut: null
onbeforepaste: null
onblur: null
oncancel: null
oncanplay: null
oncanplaythrough: null
onchange: null
onclick: null
onclose: null
oncontextmenu: null
oncopy: null
oncuechange: null
oncut: null
ondblclick: null
ondrag: null
ondragend: null
ondragenter: null
ondragleave: null
```

```
ondragover: null
ondragstart: null
ondrop: null
ondurationchange: null
onemptied: null
onended: null
onerror: null
onfocus: null
oninput: null
oninvalid: null
onkeydown: null
onkeypress: null
onkeyup: null
onload: null
onloadeddata: null
onloadedmetadata: null
onloadstart: null
onmousedown: null
onmouseenter: null
onmouseleave: null
onmousemove: null
onmouseout: null
onmouseover: null
onmouseup: null
onmousewheel: null
onpaste: null
onpause: null
onplay: null
onplaying: null
onprogress: null
onratechange: null
onreset: null
onresize: null
onscroll: null
onsearch: null
onseeked: null
onseeking: null
onselect: null
onselectstart: null
onshow: null
```

```
onstalled: null
onsubmit: null
onsuspend: null
ontimeupdate: null
ontoggle: null
onvolumechange: null
onwaiting: null
onwebkitfullscreenchange: null
onwebkitfullscreenerror: null
onwheel: null
outerHTML: "<p id="target">Hello, World!</p>"
outerText: "Hello, World!"
ownerDocument: document
parentElement: null
parentNode: null
prefix: null
previousElementSibling: null
previousSibling: null
scrollHeight: 0
scrollLeft: 0
scrollTop: 0
scrollWidth: 0
shadowRoot: null
spellcheck: true
style: CSSStyle
DeclarationtabIndex: -1
tagName: "P"
textContent: "Hello, World!"
title: ""
translate: true
webkitdropzone: ""
__proto__: HTMLParagraphElement
```

通过使用 DOM 提供的 API (Application Program Interface) 可以动态的修改节点（node），也就是对 DOM 树的直接操作。浏览器中通过使用 JavaScript 来实现对于 DOM 树的改动。

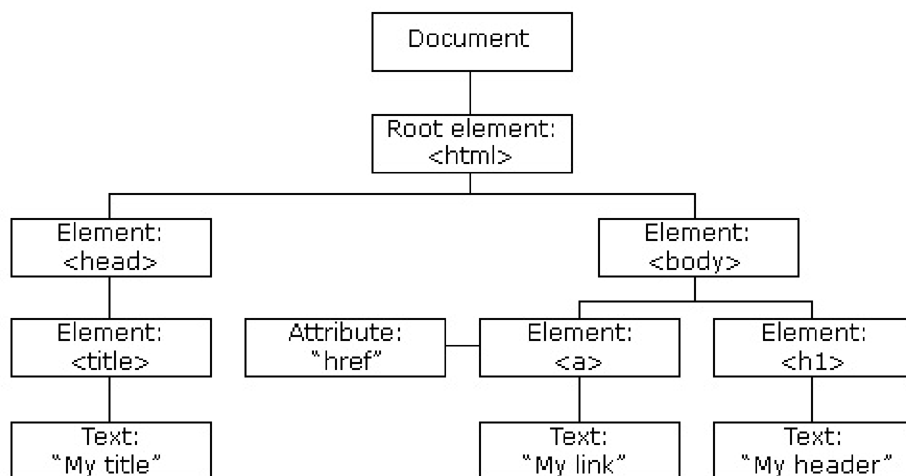
DOM 包含

- DOM Core

- DOM HTML
- DOM Style
- DOM Event

HTML 转换 DOM 树

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>My title</title>
  </head>
  <body>
    <a href="">My Link</a>
    <h1>My header</h1>
  </body>
</html>
```



节点遍历

在元素节点中提取自己所需的节点，并予以操作。

```
// Document.getElementsByTagName()
// 更具标签名找到目标节点的集合，此例中为 <h1>My header</h1>
var node = document.getElementsByTagName('h1')[0];

// Node.parentNode;
// 获得目标节点的父节点，此例中为 body 元素
node.parentNode;

// Node.firstChild
// 获得目标节点的第一个子节点，此例中为 "My header"
node.firstChild;

// Node.lastChild
// 获得目标节点的最后一个子节点，此例中为 "My header"
node.lastChild;

// Node.previousSibling;
// 获得目标节点的前一个相邻节点
node.previousSibling;

// Node.nextSibling;
// 获得目标节点的下一个相邻节点
node.nextSibling;
```

节点类型

常用节点类型

- ELEMENT_NODE 可使用 `Document.createElement('elementName');` 创建
- TEXT_NODE 可使用 `Document.createTextNode('Text Value');` 创建

不常用节点类型

- COMMENT_NODE
- DOCUMENT_TYPE_NODE

不同节点对应的**NodeType**类型

此值可以通过 `Node.nodeType` 来获取。

节点编号	节点名称
1	Element
2	Attribute
3	Text
4	CDATA Section
5	Entity Reference
6	Entity
7	Processing Instrucion
8	Comment
9	Document
10	Document Type
11	Document Fragment
12	Notation

NOTE：此处需要清楚 `节点` 和 `元素` 的区别。我们平常说的 `元素` 其实指的是节点中得 `元素节点` ，所以说 `节点` 包含 `元素` ，节点还包括文本节点、实体节点等。

元素遍历

元素节点符合 HTML DOM 树规则，所以它与 DOM 中存在的节点相似。

```
<p>
  Hello,
    <em>Xinyang</em>!
  回到
    <a href="http://li-xinyang.com">
      主页
    </a>
  。
</p>
```

```
// 在选取元素节点后

p.firstElementChild;       // <em>Xinyang</em>
p.lastElementChild;        // <a href="http://li-xinyang.com">主
页</a>

em.nextElementSibling;     // <a href="http://li-xinyang.com">主
页</a>
em.previousElementSibling; // "Hello,"
```

Table of Contents *generated with [DocToc]*

节点操作

因为 DOM 的存在，这使我们可以通过 JavaScript 来获取、创建、修改、或删除节点。

NOTE：下面提供的例子中的 `element` 均为元素节点。

获取节点

父子关系

- `element.parentNode`
- `element.firstChild` / `element.lastChild`
- `element.childNodes` / `element.children`

兄弟关系

- `element.previousSibling` / `element.nextSibling`
- `element.previousElementSibling` / `element.nextElementSibling`

通过节点直接的关系获取节点会导致代码维护性大大降低（节点之间的关系变化会直接影响到获取节点），而通过接口则可以有效的解决此问题。

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>ELEMENT_NODE & TEXT_NODE</title>
</head>
<body>
  <ul id="ul">
    <li>First</li>
    <li>Second</li>
    <li>Third</li>
    <li>Fourth</li>
  </ul>
  <p>Hello</p>
  <script type="text/javascript">
    var ulNode = document.getElementsByTagName("ul")[0];
    console.log(ulNode.parentNode);             //<body></body>
    console.log(ulNode.previousElementSibling); //null
    console.log(ulNode.nextElementSibling);     //<p>Hello</p>
    console.log(ulNode.firstElementChild);      //<li>First</li>
    console.log(ulNode.lastElementChild);       //<li>Fourth</li>

  </script>
</body>
</html>
```

NTOE：细心地人会发现，在节点遍历的例子中，body、ul、li、p节点之间是没有空格的，因为如果有空格，那么空格就会被当做一个TEXT节点，从而用ulNode.previousSibling获取到得就是一个空的文本节点，而不是 `<li>First</li>` 节点了。即节点遍历的几个属性会得到所有的节点类型，而元素遍历只会得到相对应的元素节点。一般情况下，用得比较多得还是元素节点的遍历属性。

实现浏览器兼容版的**element.children**

有一些低版本的浏览器并不支持 `element.children` 方法，但我们可以用下面的方式来实现兼容。

```
<html lang>
<head>
  <meta charest="utf-8">
  <title>Compatible Children Method</title>
</head>
<body id="body">
  <div id="item">
    <div>123</div>
    <p>ppp</p>
    <h1>h1</h1>
  </div>
  <script type="text/javascript">
    function getElementChildren(e){
      if(e.children){
        return e.children;
      }else{
        /* compatible other browse */
        var i, len, children = [];
        var child = element.firstChild;
        if(child != element.lastChild){
          while(child != null){
            if(child.nodeType == 1){
              children.push(child);
            }
            child = child.nextSibling;
          }
        }else{
          children.push(child);
        }
        return children;
      }
    }
    /* Test method getElementChildren(e) */
    var item = document.getElementById("item");
    var children = getElementChildren(item);
    for(var i =0; i < children.length; i++){
      alert(children[i]);
```

```
    }
  </script>
</body>
</html>
```

NOTE：此兼容方法为初稿，还未进行兼容性测试。

接口获取元素节点

- `getElementById`
- `getElementsByTagName`
- `getElementsByClassName`
- `querySelector`
- `querySelectorAll`

API	只作用于 `document`	唯一返回值	live
getElementById	√	√	
getElementsByTagName			√
getElementsByClassName			√
querySelectorAll			
querySelector		√	

getElementById

获取文档中指定 `id` 的节点对象。

```
var element = document.getElementById('id');
```

getElementsByTagName

动态的获取具有指定标签元素节点的集合（其返回值会被 DOM 的变化所影响，其值会发生变化）。此接口可直接通过元素而获取，不必直接作用于 `document` 之上。

```
// 示例
var collection = element.getElementsByTagName('tagName');

// 获取指定元素的所有节点
var allNodes = document.getElementsByTagName('*');

// 获取所有 p 元素的节点
var elements = document.getElementsByTagName('p');
// 取出第一个 p 元素
var p = elements[0];
```

getElementsByClassName

获取指定元素中具有指定 `class` 的所有节点。多个 `class` 可的选择可使用空格分隔，与顺序无关。

```
var elements = element.getElementsByClassName('className');
```

NOTE：IE9 及一下版本不支持 `getElementsByClassName`

兼容方法

```
function getElementsByClassName(root, className) {
  // 特性侦测
  if (root.getElementsByClassName) {
    // 优先使用 W3C 规范接口
    return root.getElementsByClassName(className);
  } else {
    // 获取所有后代节点
    var elements = root.getElementsByTagName('*');
    var result = [];
    var element = null;
    var classNameStr = null;
    var flag = null;

    className = className.split(' ');

    // 选择包含 class 的元素
    for (var i = 0, element; element = elements[i]; i++) {
      classNameStr = ' ' + element.getAttribute('class') + ' ';
      flag = true;
      for (var j = 0, name; name = className[j]; j++) {
        if (classNameStr.indexOf(' ' + name + ' ') === -1) {
          flag = false;
          break;
        }
      }
      if (flag) {
        result.push(element);
      }
    }
    return result;
  }
}
```

querySelector / querySelectorAll

获取一个 `list` （其返回结果不会被之后 DOM 的修改所影响，获取后不会再变化）符合传入的 CSS 选择器的第一个元素或全部元素。

```
var listElementNode = element.querySelector('selector');
var listElementsNodes = element.querySelectorAll('selector');

var sampleSingleNode = element.querySelector('#className');
var sampleAllNodes = element.querySelectorAll('#className');
```

NOTE: IE9 一下不支持 `querySelector` 与 `querySelectorAll`

创建节点

创建节点 -> 设置属性 -> 插入节点

```
var element = document.createElement('tagName');
```

修改节点

textContent

获取或设置节点以及其后代节点的文本内容（对于节点中的所有文本内容）。

```
element.textContent; // 获取
element.textContent = 'New Content';
```

NOTE：不支持 IE 9 及其一下版本。

innerText （不符合 W3C 规范）

获取或设置节点以及节点后代的文本内容。其作用于 `textContent` 几乎一致。

```
element.innerText;
```

NOTE：不符合 W3C 规范，不支持 FireFox 浏览器。

FireFox 兼容方案

```
if (!('innerText' in document.body)) {
  HTMLElement.prototype.__defineGetter__('innerText', function()
{
    return this.textContent;
  });
  HTMLElement.prototype.__defineSetter__('innerText', function(s
) {
    return this.textContent = s;
  });
}
```

插入节点

appendChild

在指定的元素内追加一个元素节点。

```
var aChild = element.appendChild(aChild);
```

insertBefore

在指定元素的指定节点前插入指定的元素。

```
var aChild = element.insertBefore(aChild, referenceChild);
```

删除节点

删除指定的节点的子元素节点。

```
var child = element.removeChild(child);
```

innerHTML

获取或设置指定节点之中所有的 HTML 内容。替换之前内部所有的内容并创建全新的一批节点（去除之前添加的事件和样式）。`innerHTML` 不检查内容，直接运行并替换原先的内容。

NOTE：只建议在创建全新的节点时使用。不可在用户可控的情况下使用。

```
var elementsHTML = element.innerHTML;
```

存在的问题

- 低版本 IE 存在内存泄露
- 安全问题（用户可以在名称中运行脚本代码）

Table of Contents *generated with DocToc*

属性操作

HTML 属性与 DOM 属性的对应

每个 HTML 属性都会对应相应的 DOM 对象属性。

```
<div>
  <label for="username">User Name: </label>
  <input type="input" name="username" id="username" class="text" value="">
</div>
```

```
input.id;        // 'username'
input.type;      // 'text'
input.className; // 'text'

label.htmlFor;   // 'username'
```

属性操作方式

Property Accessor

通过属性方法符得到的属性为转换过的实例对象（并非全字符串）。

特点

- X 通用行差（命名异常，使用不同的命名方式进行访问）

- X 扩展性差
- √ 实用对象（取出后可直接使用）

读取属性

```
<div>
  <label for="username">User Name: </label>
  <input type="input" name="username" id="username" class="text"
value="">
</div>
```

```
input.className; // 'text'
input[id];         // 'username'
```

写入属性

可增加新的属性或改写已有属性。

```
input.value = 'new value';
input[id] = 'new-id';
```

getAttribute / setAttribute

特点

- X 仅可获取字符串（使用时需转换）
- √ 通用性强

读取属性

获取到的均为属性的字符串。

```
var attribtue = element.getAttribute('attributeName');
```

写入属性

可增加新的属性或改写已有属性。

```
element.setAttribute('attributeName', value);
```

dataset

自定义属性，其为 `HTMLElement` 上的属性也是 `data-*` 的属性集。主要用于在元素上保存数据。获取的均为属性字符串。数据通常使用 AJAX 获取并存储在节点之上。

```
<div id='user' data-id='1234' data-username='x' data-email='mail
@gmail.com'></div>
```

```
div.dataset.id;          // '1234'
div.dataset.username;    // 'x'
div.dataset.email;       // 'mail@gmail.com'
```

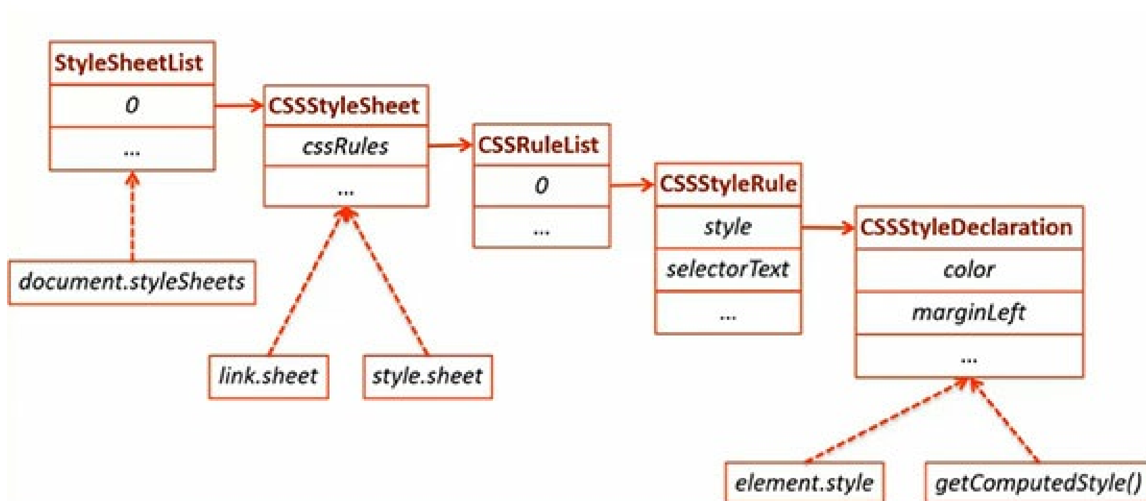
NOTE： `dataset` 在低版本 IE 不可使用，但可通过 `getAttribute` 与 `setAttribute` 来做兼容。

Table of Contents *generated with DocToc*

样式操作

通过 JavaScript 动态修改页面样式。

CSS 对应 DOM 对象



```
<link rel="stylesheet" type="text/css" href="sample.css">
// var element = document.querySelector('link');
// 对应于 element.sheet

<style type="text/css" media="screen">
  body {
    margin: 30px
  }
</style>
// var element = document.querySelector('style');
// 对应于 element.sheet

// 整个页面的全部样式（不包括行内样式）
document.styleSheets

<p style="color:red">Text Color</p>
// var element = document.querySelector('p');
// 对应于 element.style
```

内部样式表

```
<style>
  body{margin:30;}
  p{color: #aaa; line-height:20px}
</style>

// 1.对应所有样式的列表
// body{margin:30;}
// p{color: #aaa; line-height:20px}
element.sheet.cssRules;

// 2.对应相应的 CSS 选择器
// p
element.sheet.cssRules[1].selectorText;

// 3.对应一个样式
// p{color: #aaa; line-height:20px}
element.sheet.cssRules[1]

// 4.对应所有样式的键值对
// color: #aaa; line-height:20px
element.sheet.cssRules[1].style;

// 5.对应的属性值
// #aaa
element.sheet.cssRules[1].stlye.color;
element.sheet.cssRules[1].lineHeight;
```

行内样式

其对应于 `CSSStyleDeclaration` 的对象。

```
element.style.color;
// 获取行内样式的键值对
```

更新样式

element.style

```
element.style.color = 'red';
element.style.background = 'black';
```

增加样式后得到的结果

```
<div style="color: red; background: black;"></div>
```

缺点

- 每个属性的更新都需要一个命令
- 命名异常（以驼峰命名法命名属性）

element.style.cssText

一次同时设置多个行内样式，其结果同 `element.style` 单独设置相同。

```
element.style.cssText = 'color: red; background: black';
```

增加样式后得到的结果

```
<div style="color: red; background: black;"></div>
```

以上两种方式均将样式混合在逻辑当中。

更新 class

首先需要创建对应样式的 CSS 样式。

```
.angry {
  color: red;
  background: black;
}
```

然后再在 JavaScript 中，在对应的事件中给元素添加需要的类即可。

```
element.className += ' angry';
```

增加样式后得到的结果

```
<div class="angry"></div>
```

统一更新多个元素样式

以上方法均不适合同时更新多个样式，通过更换样式表的方式则可同时更改多个页面中的样式。将需要的大量样式也在一个皮肤样式表中，通过 JavaScript 来直接更换样式表来进行样式改变。（此方法也可用于批量删除样式）

```
<link rel="stylesheet" type="text/css" href="base.css">
<link rel="stylesheet" type="text/css" href="style1.css">
```

```
element.setAttribute('href', 'style2.css');
```

获取样式

element.style

其对应的为元素的行内样式表而不是实际样式表。

```
<input type="checkbox" name="" value="">
```

```
element.style.color; // ""
```

line-height: 200px

window.getComputedStyle()

将需要取出样式的目标元素传入 `window.getComputedStyle()` 函数中，即可得到对应元素的实际样式。注意的是这里获取到的样式值为只读属性不可修改！

NOTE：获取的实际为 `CSSStyleDeclaration` 的实例对象。 NOTE+：此方法不支持 IE9 以下版本，IE9 中需使用 `element.currentStyle` 来做兼容。

```
var style = window.getComputedStyle(element[, pseudoEle]);
```

```
<input type="checkbox" name="" value="">
```

```
window.getComputedStyle(element).color; // 'rgb(0,0,0)'
```

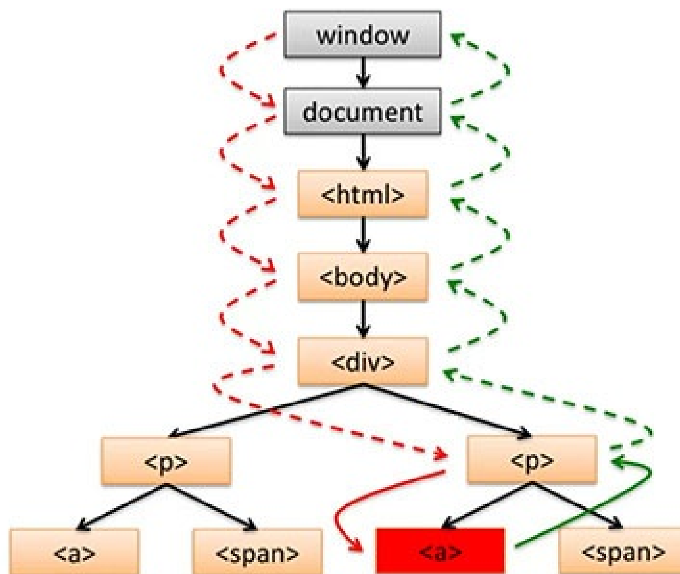
Table of Contents *generated with DocToc*

DOM 事件

何为 DOM 事件，HTML DOM 使JavaScript 有能力对 HTML 事件做出反应。（例如，点击 DOM 元素，键盘被按，输入框输入内容以及页面加载完毕等）

事件流

一个 DOM 事件可以分为 `捕获过程` 、 `触发过程` 、 `冒泡过程` 。 DOM 事件流为 DOM 事件的处理及执行的过程。下面以一个 `<a>` 元素被点击为例。

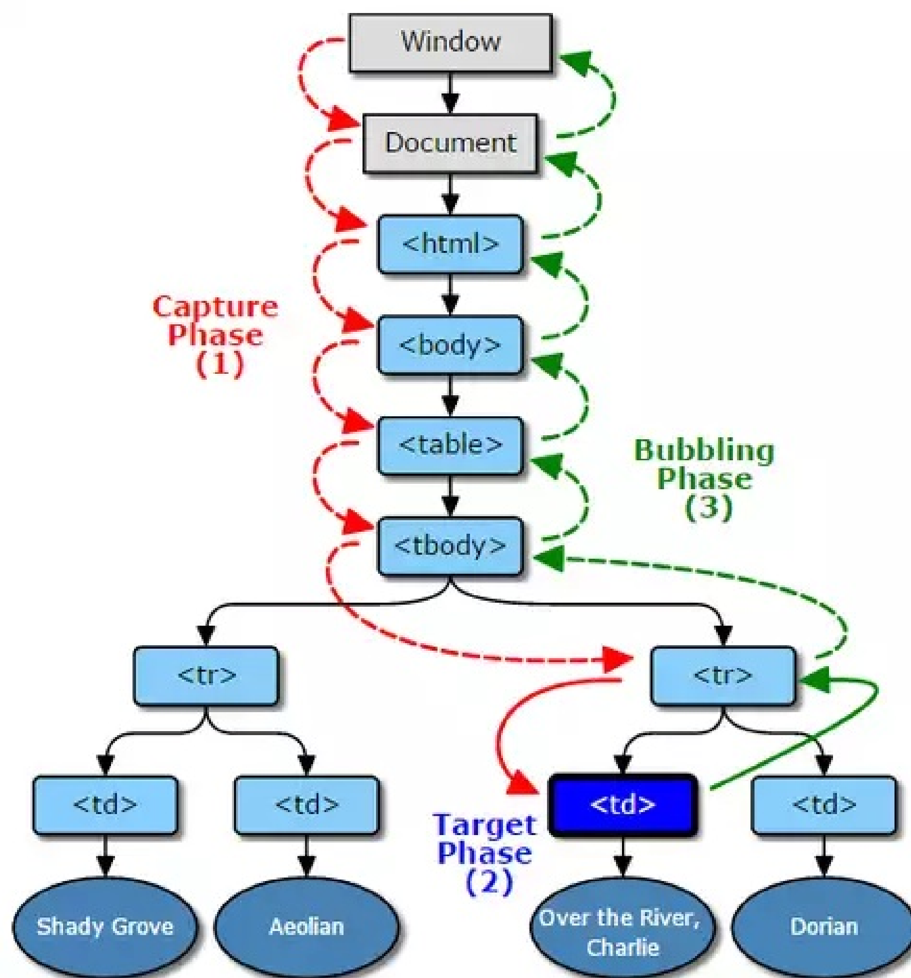


1. [红虚线]Capture Phase（事件捕获过程）当 DOM 事件发生时，它会从window节点一路跑下去直到触发事件元素的父节点为止，去捕获触发事件的元素。
2. [红绿实线]Target Phase（事件触发过程）当事件被捕获之后就开始执行事件绑定的代码
3. [绿虚线]Bubble Phase（冒泡过程）当事件代码执行完毕后，浏览器会从触发事件元素的父节点开始一直冒泡到window元素（即元素的祖先元素也会触发这个元素所触发的事件）

关于捕获过程的补充

如果有一个支持三个阶段的事件，它一定在触发时遵循下面的顺序：

```
Capture -> Target -> Bubbling
```


Window
Document
<html>
<body>
<table>
<tbody>
<tr>
<tr>
<td>
<td>
<td>
<td>
Shady Grove
Aeolian
Over the River, Charlie
Dorian
Capture Phase (1)
Bubbling Phase (3)
Target Phase (2)

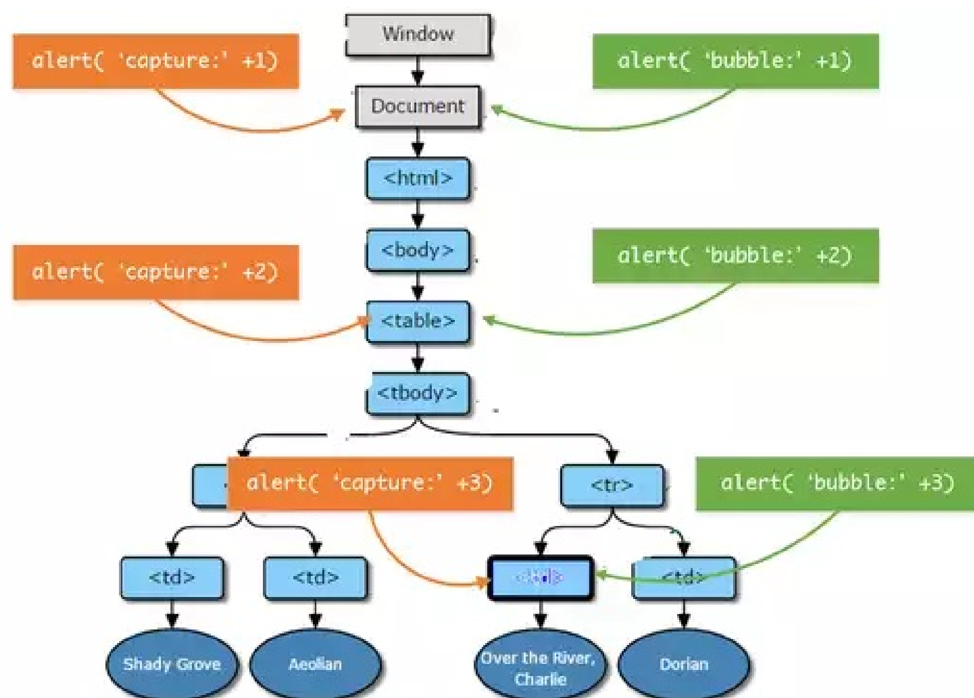
使用下面的代码来举例：

```
// 添加Capture阶段事件
docuemnt.addEventListener('click',function(){
    alert('capture：'+1);
},true);
tableNode.addEventListener('click',function(){
    alert('capture：'+2);
},true);
tdNode.addEventListener('click',function(){
    alert('capture：'+3);
},true);

// 添加Bubbling阶段事件
docuemnt.addEventListener('click',function(){
    alert('bubble：'+1);
});
tableNode.addEventListener('click',function(){
    alert('bubble：'+2);
});
tdNode.addEventListener('click',function(){
    alert('bubble：'+3);
});
```

输出结果为：

```
capture：1
capture：2
capture：3
bubble：3
bubble：2
bubble：1
```




```
// 对document添加了三个bubbling阶段的事件
document.addEventListener('click',function(){
    alert(1);
});
document.addEventListener('click',function(){
    alert(2);
});
document.addEventListener('click',function(){
    alert(3);
});
```

如上面的代码所示，其为同一节点添加了同一阶段的多个事件，那执行顺序如何呢？早期并没有规范定义，DOM 3 中规范已经明确规定 同一节点同一阶段的事件应按照注册函数的顺序执行。

> 在实际项目过程中，某些情况下比如若干的组件或者模块都需要监听某个节点的某个事件，但是组件或者模块的生成（即添加事件的时机）是不一定保证顺序的，所以这个情况下如果某个组件对这个节点的这个事件的优先级特别高（需要保证必须先触发这个组件里的这个事件）而这个平台又支持这个阶段事件的话可以添加 capture 阶段事件，用三阶段的顺序来保证，比如移动平台模拟手势的实现会添加 document 上 touchXXX 的 capture 阶段事件，以优先识别手势操作 当然实践过程中考虑到不同浏览器对三阶段支持的情况的差异，大部分情况下都采用的是 bubbling 阶段的事件
>
> —— 蔡剑飞 网易前端工程师

NOTE：低版本 IE 中并未实现捕获过程。也不是所有事件均存在这三个完整的过程（例如 `load` 没有冒泡事件）

NOTE+：在这三个阶段中无论将事件捕获和事件处理注册到任意一个父或祖父节点上都会被触发事件。

事件注册

事件注册，取消以及触发其作用对象均为一个 DOM 元素。

注册事件

```
eventTarget.addEventListener(type, listener[,useCapture])
```

- evenTarget 表示要绑定事件的DOM元素
- type 表示要绑定的事件，如："click"
- listener 表示要绑定的函数
- useCapture 可选参数，表示是否捕获过程

NOTE： `useCapture` 为设定是否为捕获过程，默认事件均为冒泡过程，只有 `useCapture` 为 `true` 时才会启用捕获过程。

```
// 获取元素
var elem = document.getElemenyById('id');

// 事件处理函数
var clickHandler = function(event) {
  // statements
};

// 注册事件
elem.addEventListener('click', clickHandler, false);

// 第二种方式，不建议使用
elem.onclick = clickHandler;
// 或者来弥补只可触发一个处理函数的缺陷
elem.onclick = function(){
  clickHandler();
  func();
  // 其他处理函数
};
```

取消事件

```
eventTarget.removeEventListener(type, listener[,useCapture]);
```

- evenTarget 表示要绑定事件的DOM元素
- type 表示要绑定的事件，如："click"
- listener 表示要绑定的函数
- useCapture 可选参数，表示是否捕获过程

```
// 获取元素
var elem = document.getElemenyById('id');

// 取消事件
elem.removeEventListener('click', clickHandler, false);

// 第二种方式。不建议使用
elem.onclick = null;
```

触发事件

点击元素，按下按键均会触发 DOM 事件，当然也可以以通过代码来触发事件。

```
eventTarget.dispatchEvent(type);

// 获取元素
var elem = document.getElemenyById('id');

// 触发事件
elem.dispatchEvent('click');
```

浏览器兼容型

以上均为 **W3C**定义的标准定义，但早期浏览器 IE8 及其以下版本，均没有采用标准的实现方式。不过这些低版本浏览器也提供了对于 DOM 事件的注册、取消以及触发的实现。

事件注册与取消，`attchEvent/detachEvent` 。事件触发，`fireEvent(e)` ，其也不存在捕获阶段（Capture Phase）。

兼容低版本代码实现

注册事件

```
var addEvent = document.addEventListener ?
  function(elem, type, listener, useCapture) {
    elem.addEventListener(type, listener, useCapture);
  } :
  function(elem, type, listener, useCapture) {
    elem.attachEvent('on' + type, listener);
  }
```

取消事件

```
var addEvent = document.removeElementListener ?
  function(elem, type, listener, useCapture) {
    elem.removeElementListener(type, listener, useCapture);
  } :
  function(elem, type, listener, useCapture) {
    elem.detachEvent('on' + type, listener);
  }
```

事件对象

调用事件处理函数时传入的信息对象，这个对象中含有关于这个事件的详细状态和信息，它就是事件对象 `event` 。其中可能包含鼠标的位置，键盘信息等。

```
// 获取元素
var elem = document.getElemenyById('id');

// 事件处理函数
var clickHandler = function(event) {
  // statements
};

// 注册事件
elem.addEventListener('click', clickHandler, false);
```

NOTE：在低版本 IE 中事件对象是被注册在 `window` 之上而非目标对象上。使用下面的兼容代码既可解决。

```
var elem = document.getElemenyById('id');

// 事件处理函数
var clickHandler = function(event) {
  event = event || window.event;
  // statements
};
```

属性和方法

通用属性和方法

属性

- type 事件类型
- target(srcElement IE 低版本) 事件触发节点
- currentTarget 处理事件的节点

方法

- stopPropagation 阻止事件冒泡传播
- preventDefault 阻止默认行为
- stopImmediatePropagation 阻止冒泡传播

阻止事件传播

`event.stopPropagation()` （W3C规范方法），如果在当前节点已经处理了事件，则可以阻止事件被冒泡传播至 DOM 树最顶端即 `window` 对象。

`event.stopImmediatePropagation()` 此方法同上面的方法类似，除了阻止将事件冒泡传播值最高的 DOM 元素外，还会阻止在此事件后的事件的触发。

`event.cancelBubble=true` 为 IE 低版本中中对于阻止冒泡传播的实现。

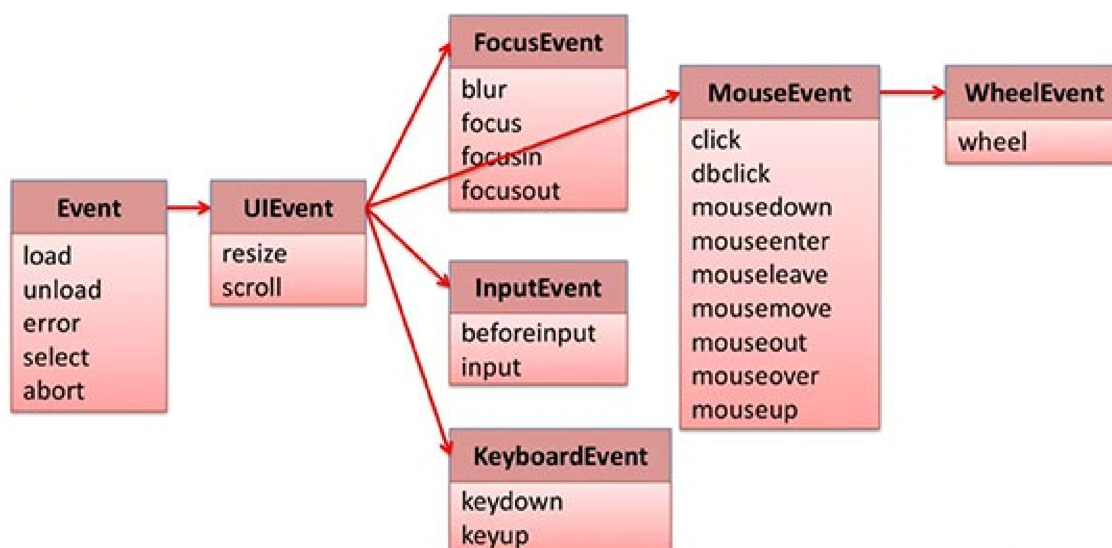
阻止默认行为

默认行为是指浏览器定义的默认行为（点击一个链接的时候，链接默认就会打开。当我们双击文字的时候，文字就会被选中），比如单击链接可以打开新窗口。

`Event.preventDefault()` 为 W3C 规范方法，在 IE 中的实现方法为 `Event.returnValue=false` 。

事件分类

Event



事件类型	是否冒泡	元素	默认事件	元素例子
load	NO	Window, Document, Element	None	window, image, iframe
unload	NO	Window, Document, Element	None	window
error	NO	Window, Element	None	window, image
select	NO	Element	None	input, textarea
abort	NO	Window, Element	None	window, image

window

- load 页面全部加载完毕
- unload 离开本页之前的卸载
- error 页面异常
- abort 取消加载

image

- load 图片加载完毕
- error 图标加载错误
- abort 取消图标加载

在目标图标不能正常载入时，载入备份替代图来提供用户体验。

```
<img src="http://sample.com/img.png" onerror="this.src='http://sample.com/default.png'">
```

UIEvent

事件类型	是否冒泡	元素	默认事件	元素例子
resize	NO	Window, Element	None	window, iframe
scroll	NO/YES	Document, Element	None	document, div

NOTE： `resize` 为改变浏览器或 `iframe` 的窗体大小时会触发事件， `scroll` 则会在滑动内容时触发，作用于 `Document` 则不会冒泡，作用于内部元素则会冒泡。

MouseEvent

DOM 事件中最常见的事件之一。

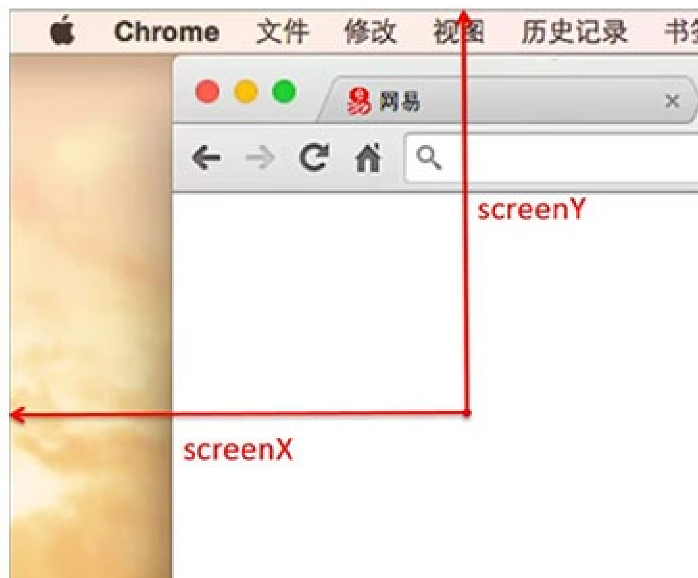
事件类型	是否冒泡	元素	默认事件	元素例子
click	YES	Element	focus/activation	div
dbclick	YES	Element	focus/activation/select	div
mousedown	YES	Element	drag/scroll/text selection	div
mosuemove	YES	Element	None	div
mouseout	YES	Element	None	div
mouseover	YES	Element	None	div
mouseup	YES	Element	context menu	div
mouseenter	NO	Element	None	div
mouseleave	NO	Element	None	div

NOTE： `mouseenter` 与 `mouseover` 的区别为前者在鼠标在子元素直接移动不会触发事件，而后者会触发。 `mouseleave` 与 `mouseout` 同上相似。

属性

- clientX, clientX
- screenX, screenY

- ctrlKey, shiftKey, altKey, metaKey 如果被按下则为真（true）
- button(0, 1, 2) 鼠标的间位



MouseEvent 顺序

鼠标的移动过程中会产生很多事件。事件的监察频率又浏览器决定。

例子：从元素 **A** 上方移动过

mousemove -> mouseover(A) -> mouseenter(A) -> mousemove(A) -> mouseout(A) -> mouseleave(A)

例子：点击元素

mousedown -> [mousemove] -> mouseup -> click

实例：拖动元素

```
<div id="div0"></div>
<style media="screen">
  #div0 {
    position: absolute;
    top: 0;
    left: 0;
    width: 100px;
    height: 100px;
    border: 1px solid black;
  }
</style>
```

```
var elem = document.getElemenyById('div0');
var clientX, clientY, isMoving;
var mouseDownHandler = function(event) {
  event = event || window.event;
  clientX = event.clientX;
  clientY = event.clientY;
  isMoving = true;
}

var mouseMoveHandler = function(event) {
  if (!isMoving) return;
  event = event || window.event;
  var newClientX = event.clientX,
      newClientY = event.clientY;
  var left = parseInt(elem.style.left) || 0,
      top = parseInt(elem.style.top) || 0;
  elem.style.left = left + (newClientX - clientX) + 'px';
  elem.style.top = top + (newClientY - clientY) + 'px';
  clientX = newClientX;
  clientY = newClientY;
}

var mouseUpHandler = function() {
  isMoving = false;
}

addEvent(elem, 'mousedown', mouseDownHandler);
addEvent(elem, 'mouseup', mouseUpHandler);
addEvent(elem, 'mousemove', mouseMoveHandler);
```

滚轮事件（**Wheel**）

事件类型	是否冒泡	元素	默认事件	元素例子
wheel	YES	Element	scroll or zoom document	div

属性

- deltaMode 鼠标滚轮偏移量的单位
- deltaX

- deltaY
- deltaZ

FocusEvent

其用于处理元素获得或失去焦点的事件。（例如输入框的可输入状态则为获得焦点，点击外部则失去焦点）

事件类型	是否冒泡	元素	默认事件	元素例子
blur	NO	Window, Element	None	window, input
focus	NO	Window, Element	None	window, input
focusin	NO	window, Element	None	window, input
focusout	NO	window, Element	None	window, input

NOTE： `blur` 失去焦点时， `focus` 获得焦点时， `focusin` 即将获得焦点， `focusout` 即将失去焦点。

属性

一个元素失去，既另一个元素获得焦点。这里的 `relatedTarget` 则为相对的那个元素。

- relatedTarget

InputEvent

输入框输入内容则会触发输入事件。

事件类型	是否冒泡	元素	默认事件	元素例子
beforeInput	YES	Element	update DOM Element	input
input	YES	Element	None	input

NOTE： `beforeInput` 为在按键按下后即将将输入字符显示之前生成的事件。

NOTE+：IE 并没有 `InputEvent` 则需使用 `onpropertychange(IE)` 来代替。

KeyboardEvent

其用于处理键盘事件。

事件类型	是否冒泡	元素	默认事件	元素例子
keydown	YES	Element	beforeInput/input/focus/blur/activation	div, input
keyup	YES	Element	None	div, input

属性

- key 按下的键字符串
- code
- ctrlKey, shiftKey, altKey, metaKey
- repeat 代表按键不松开为 true
- keyCode
- charCode
- which

事件代理

事件代理是指在父节点上（可为元素最近的父节点也可为上层的其他节点）处理子元素上触发的事件，其原理是通过事件流机制而完成的。可以通过事件对象中获取到触发事件的对象（如下所示）。

```
var elem = document.getElemenyById('id');
elem.addEventListener('click', function(event) {
  var e = event || window.event;
  var target = e.target || e.srcElement;
  // statements
});
```

优点

- 需要管理的事件处理函数更少
- 内存分配更少，更高效
- 增加与删除子节点可以不额外处理事件

缺点

- 事件管理的逻辑变的复杂（因为冒泡机制）

Table of Contents *generated with DocToc*

多媒体

HTML5 前的多媒体需要借助第三方插件，例如 Flash，但是 HTML5 将网页中的多媒体带入了新的一章。



基本用法

```
// 音频
// 指定资源类型可以帮助浏览器更快的定位解码
<audio autobuffer autoloop loop controls>
  <source src="/media/audio.mp3" type="audio/mpeg">
  <source src="/media/audio.oga">
  <source src="/media/audio.wav">
  <object type="audio/x-wav" data="/media/audio.wav" width="290"
height="45">
    <param name="src" value="/media/audio.wav">
    <param name="autoplay" value="false">
    <param name="autoStart" value="0">
    <p><a href="/media/audio.wav">Download this audio file.</a></
p>
  </object>
</audio>

// 视频
<video autobuffer autoloop loop controls width=320 height=240>
  <source src="/media/video.oga">
  <source src="/media/video.m4v">
  <object type="video/ogg" data="/media/video.oga" width="320" h
eight="240">
  <param name="src" value="/media/video.oga">
  <param name="autoplay" value="false">
  <param name="autoStart" value="0">
  <p><a href="/media/video.oga">Download this video file.</a></p>

  </object>
</video>
```

多媒体支持类型

HTML5 支持音频列表

HTML5 支持视频列表

多媒体格式兼容

测试音频兼容性。

```
var a = new Audio();
// 检测媒体类型返回
// 支持 - 'maybe' 或 'probably'
// 不支持 - ''
a.canPlayType('audio/nav');
```

HTML 属性

视频与音频的大部分属性和方法几乎相同。

属性	是否必须	默认值	备注
src	是		音频文件地址 URL
controls	否	false	显示控件
autoplay	否	false	音频就绪后自动播放
preload	否	none	可取值为 none、metadata、auto。音频在页面加载是进行加载，并预备播放。如果使用 autoplay 则忽略该属性（该属性失效）
loop	否	false	是否循环播放

控制多媒体

方法

- `load()` 加载资源
- `play()` 播放
- `pause()` 暂停播放

属性

- `playbackRate` 1为正常速度播放，大于1为快速播放最高20。
- `currentTime` 调准播放时间，以秒为单位。
- `volume` 取值范围0到1

- `muted` 眞假值

- `paused` 布尔值暂停
- `seeking` 布尔值跳转
- `ended` 布尔值播放完成
- `duration` 媒体时长数值
- `initialTime` 媒体开始时间

多媒体相关事件

- `loadstart` 开始请求媒体内容
- `loadmetadata` 媒体元数据以加载完成（时长，编码格式等）
- `canplay` 加载一些内容但可播放
- `play` 调用 `play()` 或设置 `autoplay`
- `waiting` 缓冲数据不够，暂停播放
- `playing` 正在进行播放

全部事件列表

事件列表

Web Audio API

音频 W3C 官网定义在这里

Mozilla 官方音频教程在这，以及第三方教程 1和教程 2。

Table of Contents *generated with DocToc*

Canvas

Mozilla 官方 `<canvas>` 教程在这里。

画布 `<canvas>` 的默认宽高为 300 与 150 ，但是同样可以使用 CSS 设定宽高。但因为 CSS 与 JavaScript 在渲染工程中有速度的差异，所以不建议使用 CSS 对 `<canvas>` 的宽高做设定。

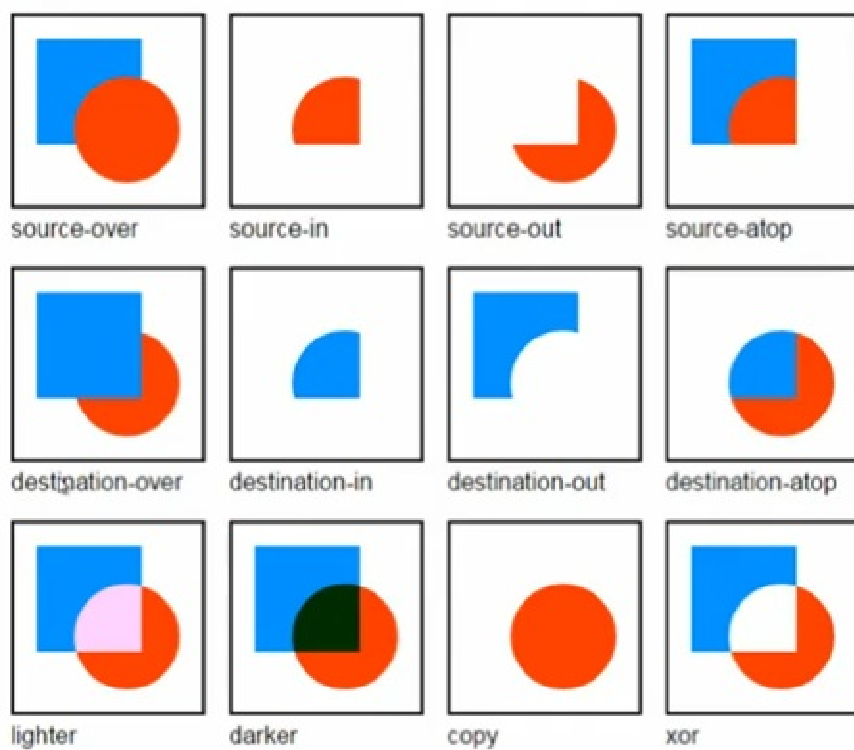
```
<canvas id="canvasId" width="300" height="150">
</canvas>
```

渲染上下文对象

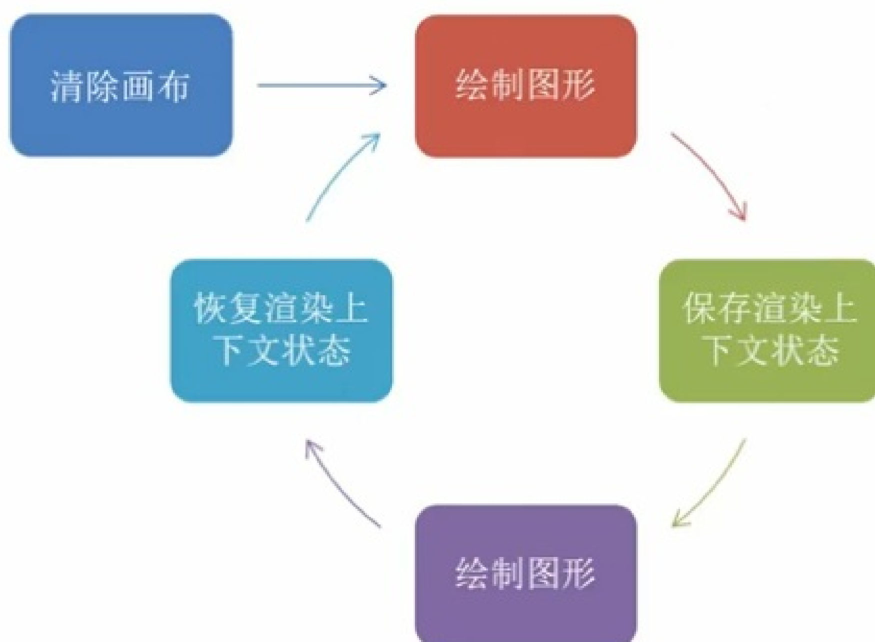
下面的 `ctx` 即为渲染上下文对象。`globalCompositeOperation` 为对于 `canvas` 绘画时的渲染属性设置，具体表现如下图所示。

```
var canvas = document.getElementById('canvasId');
var ctx = canvas.getContext('2d');


// 绘画 canvas 的属性
ctx.globalCompositeOperation = 'destination-over';
```



绘图步骤



一个周期就是完整的一帧的绘画过程。



```
var sun = new Image();
var moon = new Image();
var earth = new Image();

function init() {
  sun.src = 'Canvas_sun.png';
  moon.src = 'Canvas_moon.png';
  earth.src = 'Canvas_earth.png';
  window.requestAnimationFrame(draw);
}

function draw(){
  var ctx = document.getElementById('canvas').getContext('2d');

  ctx.globalCompositeOperation = 'destination-over';
  // 清空画布内容
  ctx.clearRect(0, 0, 300, 300);

  ctx.fillStyle = 'rgba(0, 0, 0, 0.4)';
  ctx.strokeStyle = 'rgba(0, 153, 255, 0.4)';

  // 保存画布状态
  ctx.save();
  ctx.translate(150, 150);

  // 开始绘制图形

  // 地球
  var time = new Date();
  ctx.rotate(((2*Math.PI)/60)* time.getSeconds() + ((2*Math.PI)/
```

```
60000)*time.getMilliseconds());
  ctx.translate(105, 0);
  // 阴影
  ctx.fillRect(0, -12, 50, 24);
  ctx.drawImage(earth, -12, -12);

  // 月亮
  ctx.rotate(((2*Math.PI)/6)*time.getSeconds() + ((2*Math.PI)/60
00)*time.getMilliseconds());
  ctx.translate(0, 28.5);
  ctx.drawInmage(moon, -3.5, -3.5);

  // 恢复画布状态
  ctx.restore();

  ctx.beginPath();
  ctx.arc(150, 150, 105, 0, Math.PI*2, false);

  ctx.stroke();

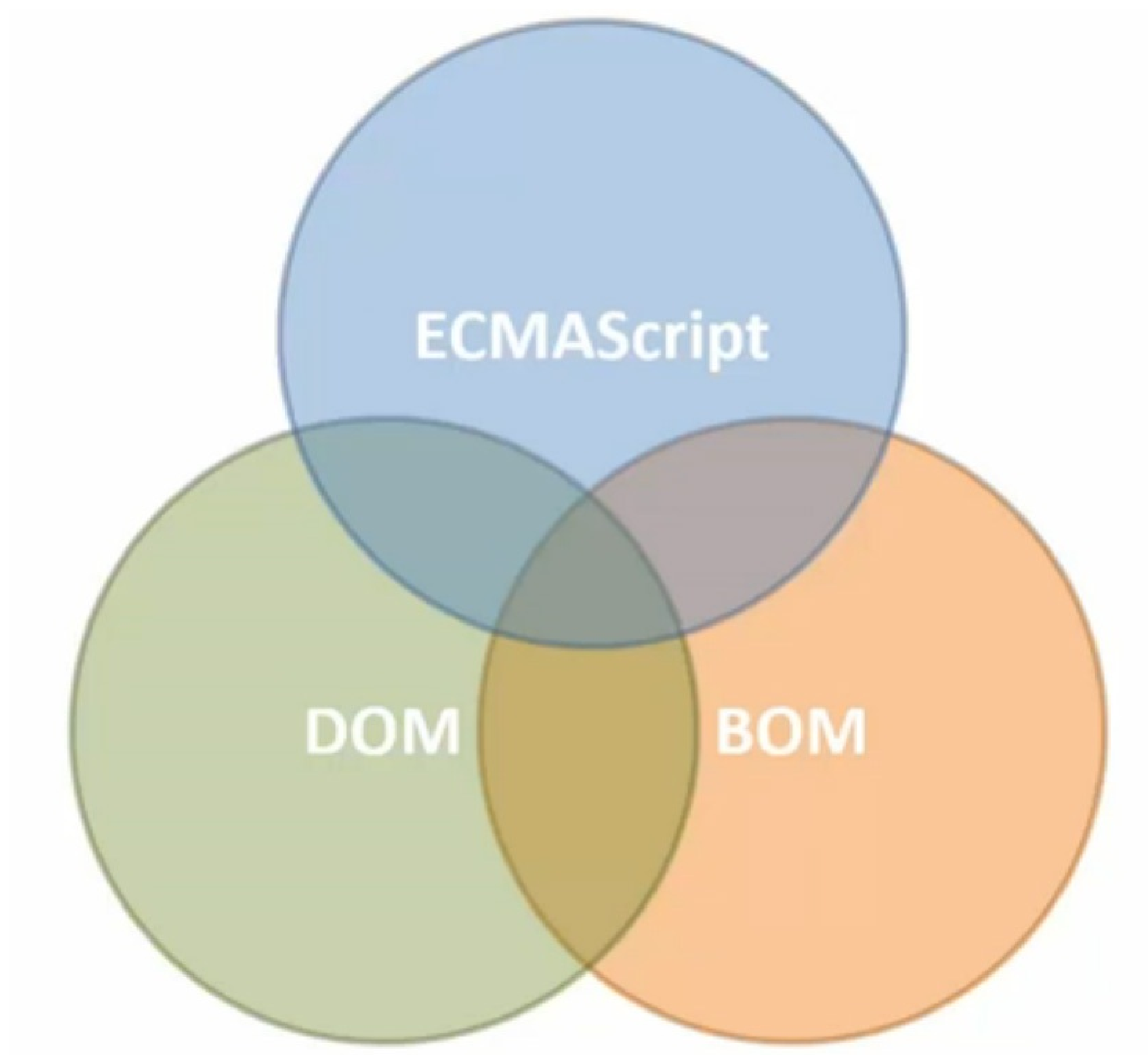
  ctx.drawImage(sun, 0, 0, 300, 300);

  window.requestAnimationFrame(draw);
}

init();
```

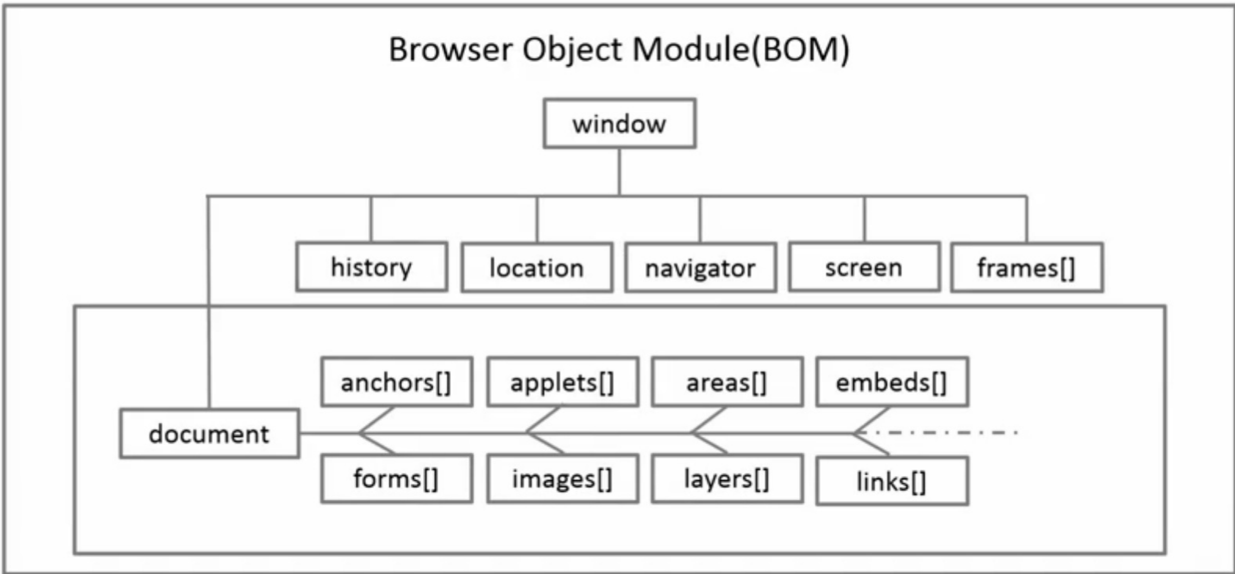
Table of Contents *generated with DocToc*

BOM



BOM 为浏览器窗口对象的一组 API。

BOM 结构图



属性

属性名	描述
navigator	浏览器信息
location	浏览器定位和导航
history	窗口浏览器历史
screen	屏幕信息

navigator

```
navigator.userAgent
```

- Chrome, Mozilla/5.0(Windows NT 6.1; WOW64) Apple WebKit/37.36 (KHTML, like Gecko) Chrome/40.0.2214.115 Safari/537.36
- Firefox, Mozilla/5.0(Windows NT 6.1; WOW64; rv:36.0) Gecko/20100101 Firefox/36.0
- IE, Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E; InfoPath.3; rv:11.0) like Gecko

```
appCodeName: "Mozilla"appName: "Netscape"appVersion: "5.0 (Macin
tosh; Intel Mac OS X 10_10_3) AppleWebKit/537.36 (KHTML, like Ge
cko) Chrome/43.0.2357.124 Safari/537.36"cookieEnabled: truedoNot
Track: "1"geolocation: GeolocationhardwareConcurrency: 8language
: "en-US"languages: Array[4]maxTouchPoints: 0mimeTypes: MimeType
ArrayonLine: truepermissions: Permissionsplatform: "MacIntel"plu
gins: PluginArrayproduct: "Gecko"productSub: "20030107"serviceWo
rker: ServiceWorkerContaineruserAgent: "Mozilla/5.0 (Macintosh;
Intel Mac OS X 10_10_3) AppleWebKit/537.36 (KHTML, like Gecko) C
hrome/43.0.2357.124 Safari/537.36"vendor: "Google Inc."vendorSub
: ""webkitPersistentStorage: DeprecatedStorageQuotawebkitTempora
ryStorage: DeprecatedStorageQuota__proto__: Navigator
```

NOTE：可以通过 `userAgent` 判断浏览器。

location

代表浏览器的定位和导航。可以使用 `location` 来操作 URL 中的各个部分。最常用的有 `href` 属性，当前访问资源的完整路径。

```
http://www.github.com:8080/index.html?user=li-xinyang&lang=zh-CN
#home
   |          |          |       |                 |
   |
protocol      |          |       |                 |
   |
         hostname     port       |                 |
   |
            \        /     pathname            search
 hash
               host
```

方法

- `assign(url)` 载入新的 url，记录浏览记录
- `replace(url)` 载入新的 url 不记录浏览记录
- `reload()` 重新载入当前页

history

浏览器当前窗口的浏览历史。

```
length: 9state: null __proto__: History
```

方法

- `back(int)` 后退
- `forward(int)` 前进
- `go(int)` 正数向前，附属向后

screen

其中包含屏幕信息。其中 `avil-` 开头的属性为可用属性，其余则为显示器设备属性。

Window 方法

方法	描述
`alert()` , `confirm()` 返回眞假, `prompt()` 返回用户输入值	三种对话框
`setTimeout()` , `setInterval()`	计时器
`open()` , `close()`	开启窗口，关闭窗口

NOTE：对话框会阻塞线程。

打开或关闭窗口

```
var w = window.open('subwindow.html', 'subwin', 'width=300, height=300, status=yes, resizable=yes');

// 既可关闭窗口
w.close();
```

NOTE：无需记忆，更多属性在使用时查询文档。

Window 事件

事件名	描述
load	文档和所有图片完成加载时
unload	离开当前文档时
beforeunload	和 unload 类似，但是它提供询问用户是否确认离开的机会
resize	拖动改变浏览器窗口大小时
scroll	拖动浏览器时

Table of Contents *generated with DocToc*

数据通信

HTTP

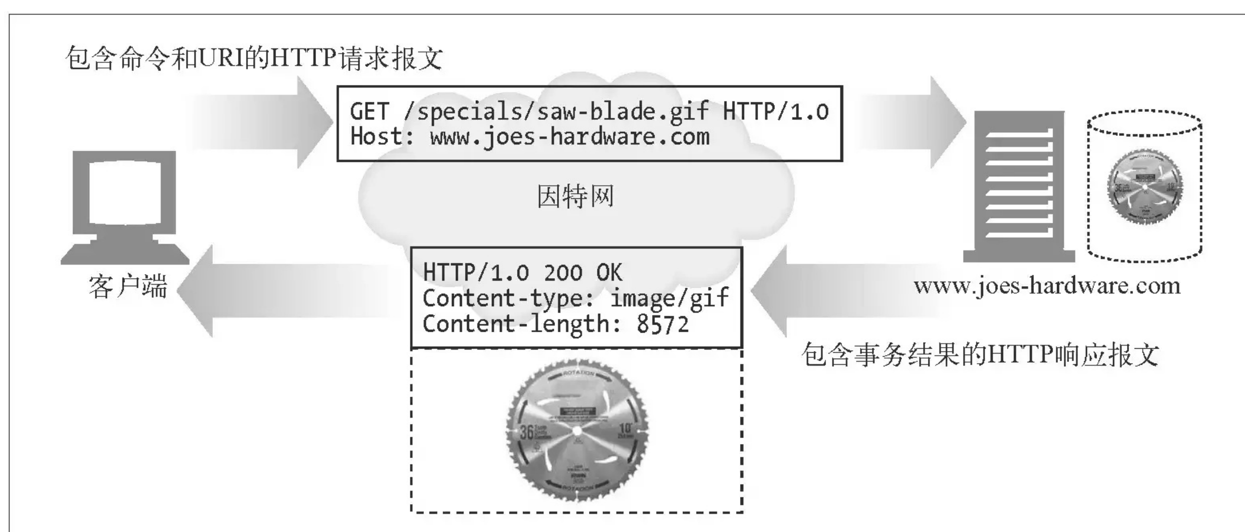
HTTP 为一个通信协议。HTTP 客户端发起请求并创建端口。HTTP 服务器在端口监听客户端的请求。 HTTP 服务器在收到请求后则返回状态和所请求的内容。

网页浏览全过程 （粗浅流程）

1. 域名解析
   i. 搜索浏览器自身 DNS 缓存
   ii. 搜索操作系统自身 DNS 缓存（如上一级未找到或已失效）
   iii. 读取本地 HOST 文件 （如上一级未找到或已失效， `/etc/hosts` ）
   iv. 浏览器发起 DNS 系统调用请求
      i. ISP 服务器查找自身缓存
      ii. ISP 服务器发起迭代（逐域寻找需要的地址）请求

2. 得到请求资源的 IP 地址
3. 发起 HTTP“三次握手”（下面为一个超级简化版）
   i. 建立连接，等待服务器确认
   ii. 服务器接受请求，回复客户
   iii. 客户端与服务器连接成功（TCP/IP 连接成功）
4. 客户根据协议发送请求
5. 服务器更具请求返回客户需求资源
6. 客户获得资源

HTTP 事务



HTTP 请求报文

GET music.163.com HTTP/1.1

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8,zh-CN;q=0.6,zh;q=0.4
Cache-Control: no-cache
Connection: keep-alive
Cookie: visited=true; playlist=65117392DNT:1
Host: music.163.com
Pragma: no-cache
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2272.118 Safari/537.36

……

其中包括主机地址，HTTP 协议版本号。头部由键值对组成。因为此请求为 GET 方法所以请求体为空。

HTTP 回复报文

```
HTTP/1.1 200 OK

Cache-Control: no-cache
Cache-Control: no-store
Connection: keep-alive
Content-Encoding: gzip
Content-Language: en-US
Content-Type: text/html;charset=utf8
Date: Fri, 17 Apr 2015 01:48:12 GMT
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Pragrma: no-cache
Server: nginx
Transfer-Encoding: chunked
Vary: Accept-Encoding

<!DOCTYPE html><html><head>......</head><body>......</body></html>
```

其中包括 HTTP 版本号，状态码及状态码描述。头部依然为键值对组成。主体则为 HTML 文件。

常用 HTTP 方法

常用方法

方法	描述	是否包含主题
GET	从服务器获取一份文档	否
POST	向服务器发送需要处理的数据	是
PUT	将请求的主题部分储存在服务器上	是
DELETE	从服务器删除一份文档	否

不常用方法

方法	描述	是否包含主题
HEAD	只从服务器获取头文档的首部	否
TRACE	对可能经过代理服务器传送到服务器上的报文进行追踪	否
OPTIONS	决定可以在服务器上执行的方法	否

URL 构成

```
http://www.github.com:8080/index.html?user=li-xinyang&lang=zh-CN
#home
   |          |          |       |                 |
   |
protocol      |          |       |                 |
   |
         hostname     port       |                 |
   |
             \        /    pathname            search
 hash
                host
```

可选部分包括：

- port
- pathname
- search
- hash

NOTE：上面提供的 URL 地址仅为参考所用。

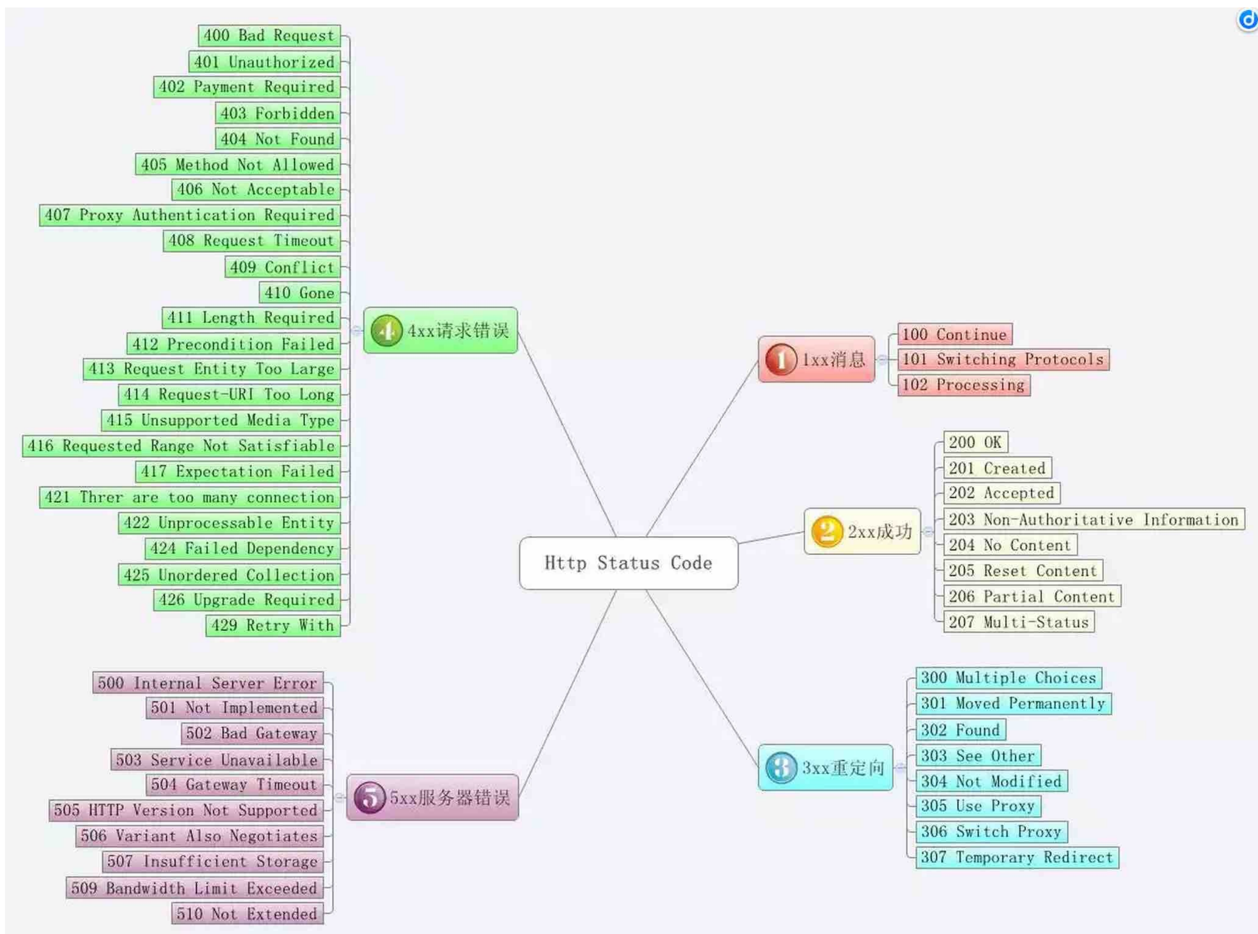
HTTP 版本

- HTTP/0.9 1991年 HTTP 原型，存在设计缺陷
- HTTP/1.0 第一个广泛应用版本
- HTTP/1.0+ 添加持久的 keep-alive 链接，虚拟主机支持，代理连接支持，成为非官方的事实版本
- HTTP/1.1 校正结构性缺陷，明确语义，引入重要的新能优化措施，删除不好的特性（当前使用版本）

NOTE：此文写于2015年6月。

常见 HTTP 状态码

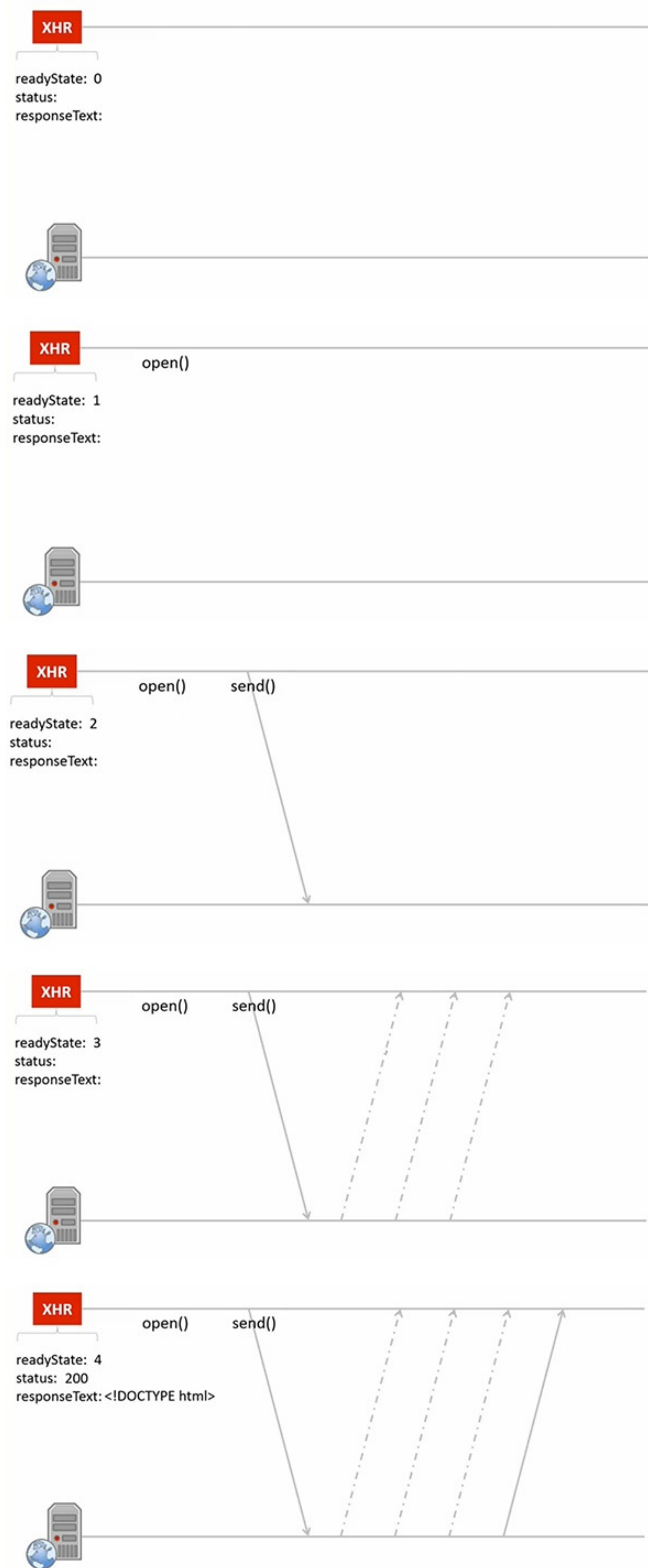
状态码	描述	代码描述
200	请求成功，一般用于 GET 和 POST 方法	OK
301	资源移动。所请求资源自动到新的 URL，浏览器自动跳转至新的 URL	Moved Permanently
304	未修改。所请求资源未修改，浏览器读取缓存数据	Not Modified
400	请求语法错误，服务器无法解析	Bad Request
404	未找到资源，可以设置个性“404页面”	Not Found
500	服务器内部错误	Internal Server Error



AJAX

AJAX（Asynchronous JavaScript and HTML）异步获取数据的概念，由 Jesse James Garrett 在2005年提出。

AJAX 请求全过程


XHR
readyState: 0
status:
responseText:
XHR
open()
readyState: 1
status:
responseText:
XHR
open()
send()
readyState: 2
status:
responseText:
XHR
open()
send()
readyState: 3
status:
responseText:
XHR
open()
send()
readyState: 4
status: 200
responseText: <!DOCTYPE html>

AJAX 调用

三部完成 AJAX 调用

1. 创建 XHR 对象
2. 处理返回数据及错误处理
3. 发送请求

```
var xhr = new XMLHttpRequest();

xhr.onreadystatechange = function(callback) {
  if (xhr.readyState === 4) {
    if ((xhr.status >== 200 && xhr.status < 300) || xhr.status =
== 304) {
      callback(xhr.responseText);
    } else {
      console.error('Request was unsuccessful: ' + xhr.status);
    }
  }
}

xhr.open('get', 'exmaple.json', true);
xhr.setRequestHeader('myHeader', 'myValue');
xhr.send(null);
```

NOTE：`xhr.onload` 只针对当 `readyState === 4` 和 `status === 200` 时的事件。

open

```
xhr.open(method, url[, async = true]);
```

- `method` 为上面说过的 HTTP 方法（例如，GET、POST）
- `url` 为资源地址
- `async` 默认为真，用于设置异步请求

setRequestHeader

```
xhr.setRequestHeader('myHeader', 'myValue');

xhr.setRequestHeader('Content-Type', 'application/x-www-form-url
encoded');
```

用于设置头部的键值对。

send

```
xhr.send([data=null]);

xhr.send()
```

数据可包含字符串或表单数控，但需要提前为 RequestHeader 做设置。

请求参数序列化

将查询参数使用字符串，跟入资源地址中。

```
xhr.open('get', 'example.json?' + 'name=value&age=value', true);
```

对象转换字符串的函数实现

```
function serialize(data) {
  if (!data) return '';
  var pairs = [];
  for (var name in data) {
    if (!data.hasOwnProperty(name)) continue;
    if (typeof data[name] === 'function') continue;
    var value = data[name].toString();
    name = encodeURIComponent(name);
    value = encodeURIComponent(value);
    pairs.push(name + '=' + value);
  }
  return pairs.join('&');
}
```

GET 请求

```
var url = 'example.json?' + serialize(formData);
xhr.open('get', url, true);
xhr.send(null);
```

POST 请求

查询参数需要作为 `send()` 的存数传入。

```
xhr.open('get', 'example.json', true);
xhr.send(serialize(formData));
```

同源策略

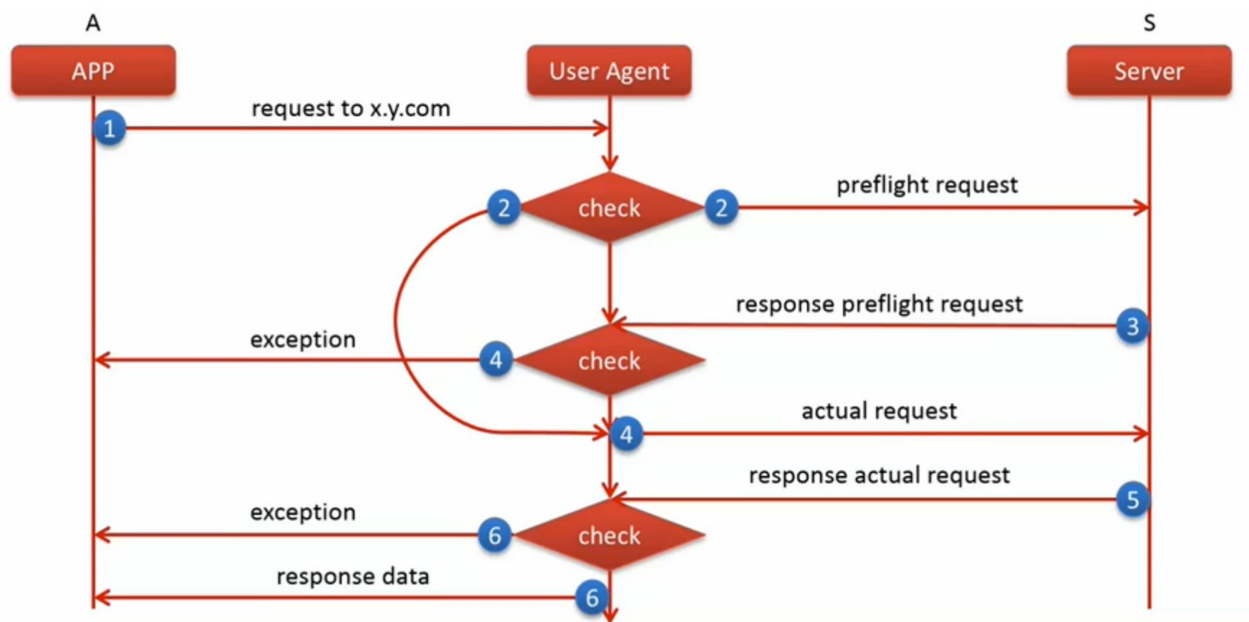
两个页面拥有相同的协议（Protocol）、端口（Port）、和主机（host）那么这两个页面就是属于同一个源（Origin）。

```
http://www.github.com:8080/index.html?user=li-xinyang&lang=zh-CN
#home
  |          |          |       |                  |
  |
protocol     |          |       |                  |
  |
         hostname     port      |                  |
  |
            \        /     pathname             search
 hash
                   host
|-----完全一致则同源------|
```

跨域资源访问

不满足同源策略的资源访问均属于跨域资源访问，W3C 定义了 **CORS**。现代浏览器已经实现了 CORS 的支持。

CORS 原理实现图

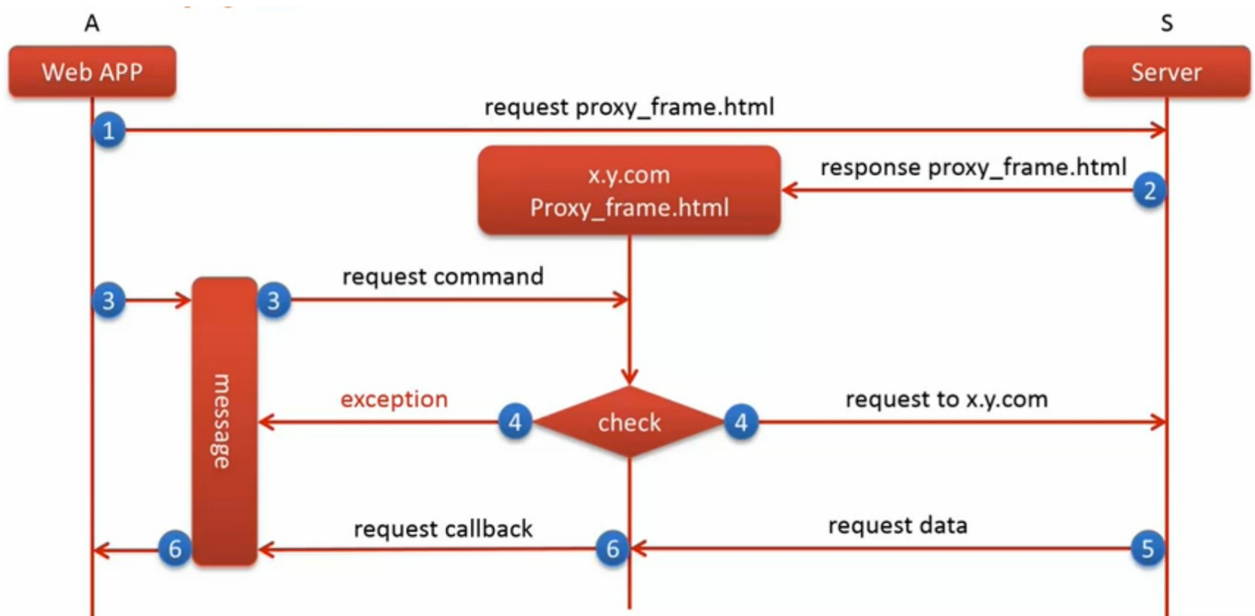


其他跨域技术

- Frame 代理
- JSONP
- Comet
- Web Sockets
- ...

Frame 代理

关于 Frame 代理的更多内容在这里。



优点：

- 参照 CORS 标准
- 支持各种请求方法 GET POST PUT DELETE

缺点：

- 需要在目标服务器防止代理理文件（造成延时）
- 低版本在大并发消息通信机制会产生延时

JSONP

全程为 JSON with Padding（填充式 JSON），它利用 `<script>` 可以跨域的原理。请求一段 JavaScript 代码，然后执行 JavaScript 代码来实现跨域。

```
function handleResponse(response) {
  alert(response.name);
}

var script = document.createElement('script');
script.src = 'http://localhost:3000/json?callback=handleResponse'
;
document.body.insertBefore(script, document.body.firstChild);
```

Table of Contents *generated with DocToc*

数据存储

Cookie

浏览器中的 Cookie 是指小型文本文件，通常在 4KB 大小左右。（由键值对构成用 `;` 隔开）大部分时候是在服务器端对 Cookie 进行设置，在头文件中 `Set-Cookie` 来对 Cookie 进行设置。

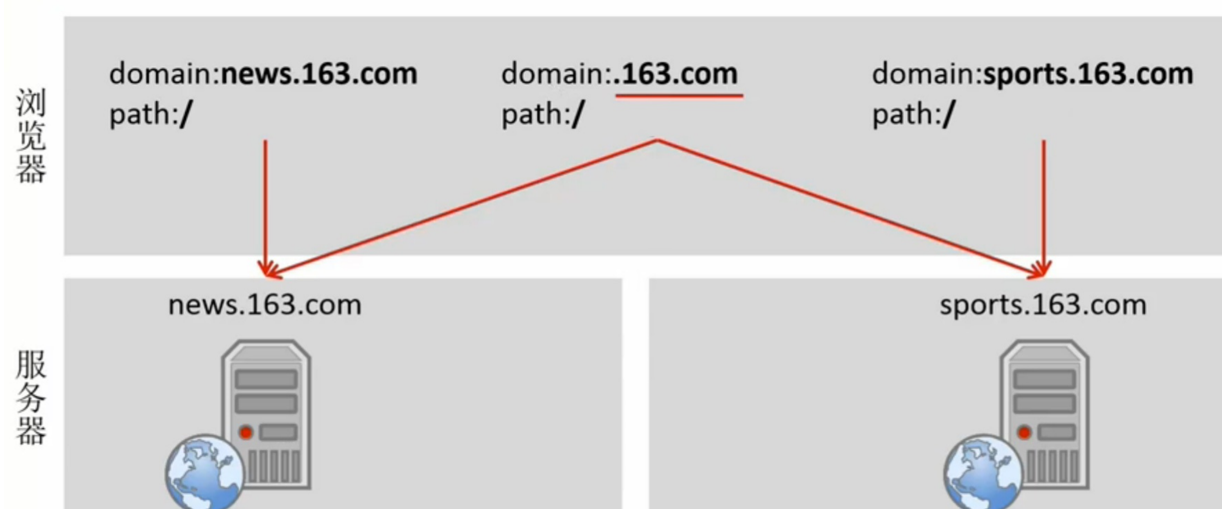
页面可以访问当前页的 Cookie 也可以访问父域的 Cookie。

属性

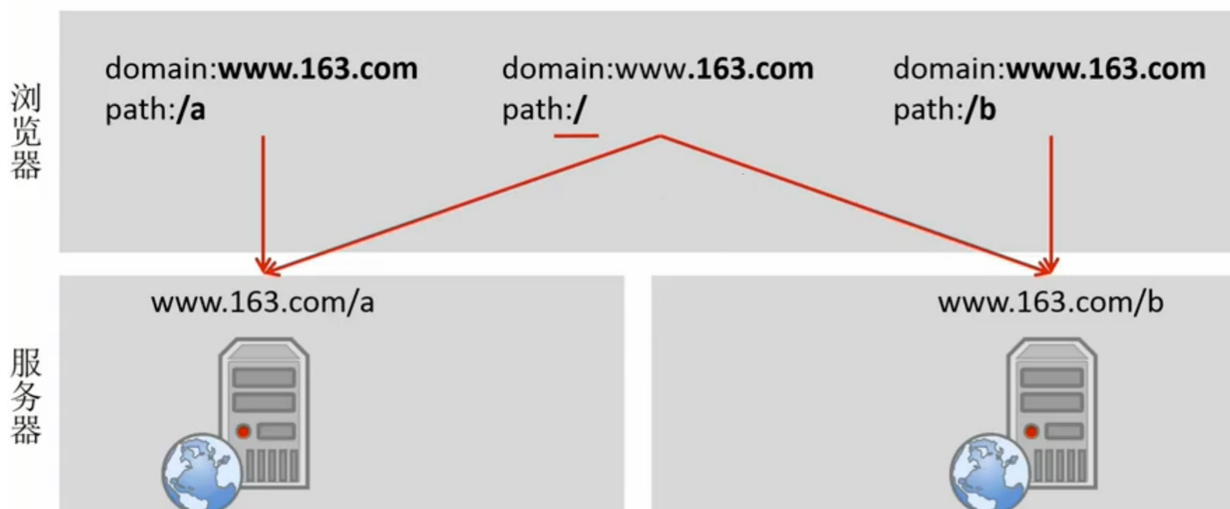
属性	默认值	作用
Name（必填）		名
Value（必填）		值
Domain	当前文档域	作用域
Path	当前文档路径	作用路径
Expires（时间戳）/Max-Age（毫秒数值）	浏览器会话时间	失效事件
Secure	false	https 协议时生效

作用域

设置作用域



设置作用路径



读取

下面转换 Cookie 至 JavaScript 对象的函数。

```
function getcookie() {
  var cookie = {};
  var all = document.cookie;
  if (all === '') return cookie;
  var list = all.split('; ');
  for (var i = 0, len = list.length; i < len; i++) {
    var item = list[i];
    var p = item.indexOf('=');
    var name = item.substring(0, p);
    name = decodeURIComponent(name);
    var value = item.substring(p + 1);
    value = decodeURIComponent(value);
    cookie[name] = value;
  }
  return cookie;
}
```

设置与修改

```
document.cookie = 'name=value';
```

下面为设置 Cookie 值的封装函数。

```
function setCookie(name, value, expires, path, domain, secure) {
  var cookie = encodeURIComponent(name) + '=' + encodeURICompone
nt(value);
  if (expires)
    cookie += '; expires=' + expires.toGMTString();
  if (path)
    cookie += '; path=' + path;
  if (domain)
    cookie += '; domain=' + domain;
  if (secure)
    cookie += '; secure=' + secure;
  document.cookie = cookie;
}
```

下面为删除** Cookie 值的函数

```
function removeCookie(name, path, domain) {
  document.cookie = 'name=' + name + '; path=' + path + '; domai
n=' + domain + '; max-age=0';
}
```

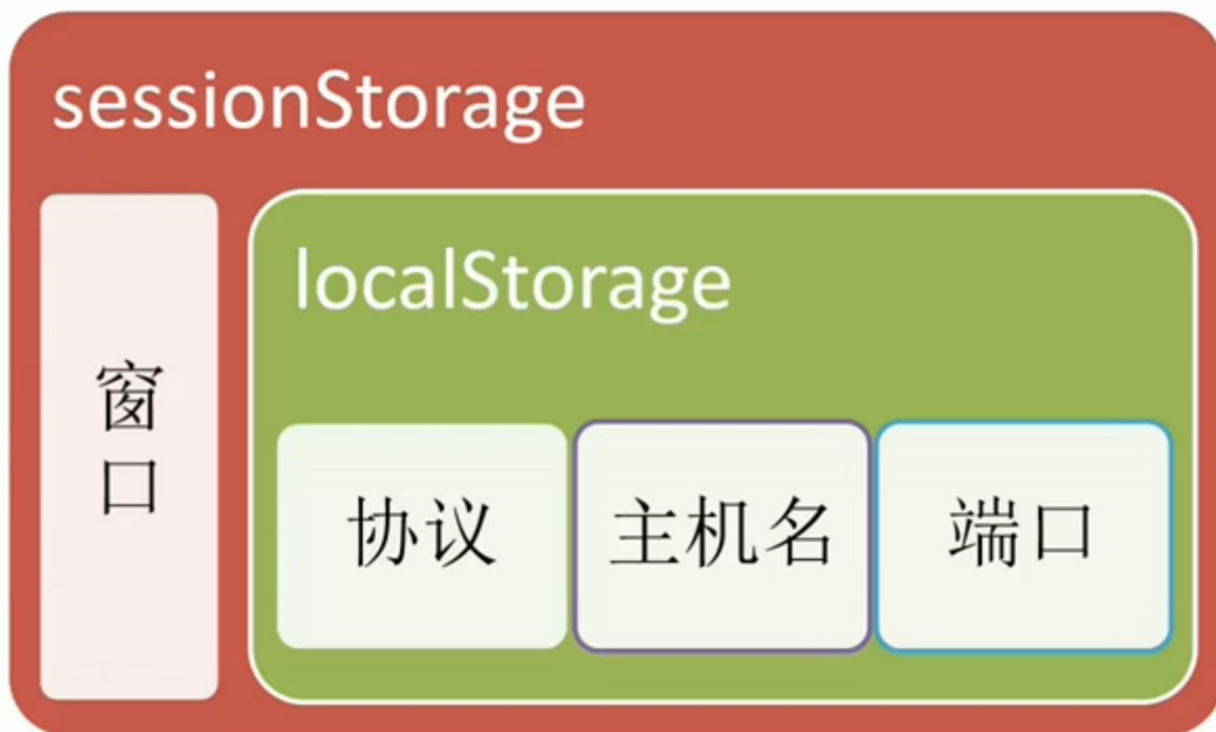
Cookie 缺陷

- 流量代价
- 安全性（明文传递）
- 大小限制

Storage

因为 Cookie 弊端的存在，所以在 HTML5 中提供了 Storage 的替代方案。

作用域的不同 Storage 分为 Local Storage 和 Session Storage，前者在用户不清理的情况下默认时间为永久，后者默认事件则为浏览器的会话时间（浏览器不同窗口直接不共享 Session Storage）。



不同浏览器对其实现的不同导致支持大小也不太，通常在 5MB 作用。

对象

读取

`localStorage.name`

添加或修改

`localStorage.name = 'Value';`

浏览器只支持字符串在 Storage 中的存储。

删除

`delete localStorage.name`

API

使用 API 操作 Storage 可以进行向下兼容的功能，在不支持的情况下可以用 Cookie 来代替。

- `localStorage.length` 获取键值对数量
- `localStorage.getItem('name')` 获取对应值
- `localStorage.key(i)` 对应值的索引获取
- `localStorage.setItem('name', 'value')` 设置键值对
- `localStorage.removeItem('name')` 删除一个值
- `localStorage.clear()` 删除所有数据

Table of Contents *generated with* *DocToc*

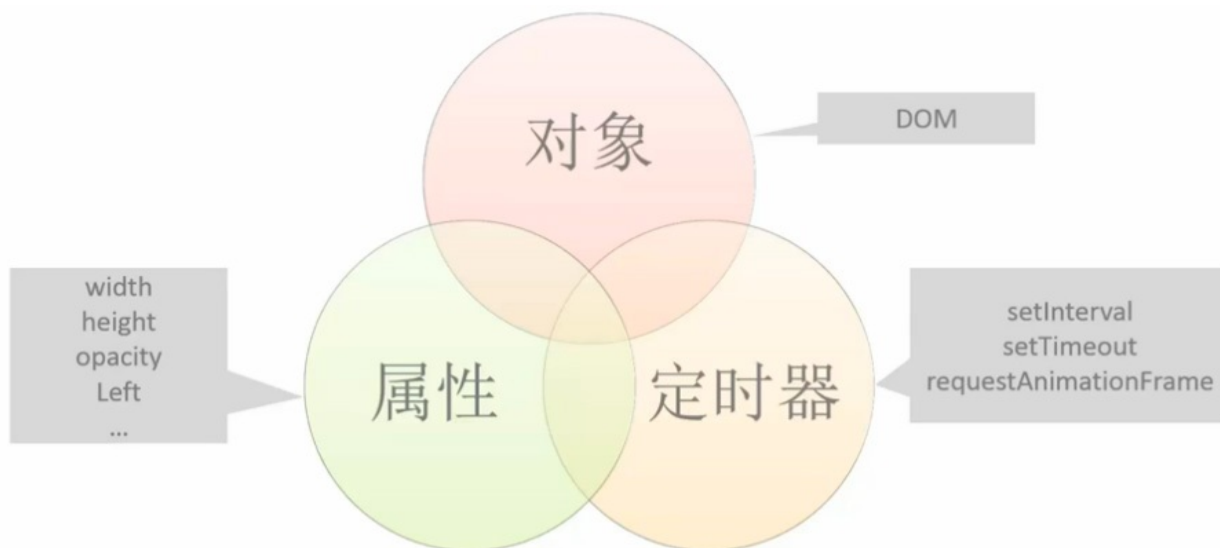
JavaScript 动画

帧，为动画的最小单位，一个静态的图像。帧频，每秒播放的帧的数量。一个动画是由很多帧组成的，因为人眼的暂留特性，当图片交替的速度大于每秒 30 帧以上既有动画的感觉。

实现方式

- gif 图像形式存储，容量大，需第三方制图工具制作。
- flash 需要第三方制作工具，不推荐。
- CSS3 实现动画具有局限性
- JavaScript 可实现大部分上面几类可实现的动画效果

JavaScript 动画三要素



定时器

setInterval

- `func` 为执行改变属性的操作
- `delay` 为出发时间间隔（毫秒为单位）
- `para1` 为执行时可传入改变属性函数的参数

```
var intervalId = setInterval(func, delay[, param1, param2, ...])
;
clearInterval(intervalId);
```

NOTE：使用 `setInterval` 可以调用一次定时器既可实现连贯的动画。使用 `clearInterval` 即可清除动画效果。

setTimeout

- `func` 为执行改变属性的操作
- `delay` 为出发时间间隔（毫秒为单位）默认为 0
- `para1` 为执行时可传入改变属性函数的参数

```
var timeoutId = setTimeout(func[, delay, param1, param2, ...]);
clearTimeout(timeoutId);
```

NOTE：使用 `setTimeout` 实现动画，则需要在动画每一帧结束时再次调用定时器。但它无需清除定时器。

区别

- `setTimeout` 在延时后只执行一次，`setInterval` 则会每隔一个延时期间后会再执行。

requestAnimationFrame

类似于 `setTimeout` 但是无需设定时间间隔。此定时器为 HTML5 中的新标准，其间隔时间不由用户控制，而是由显示器的刷新频率决定。（市面上的显示器刷新频率为每秒刷新60次）

优势

- 无需设置间隔时间
- 动画流畅度高

```
var requestId = requestAnimationFrame(func);
cancelAnimationFrame(requestId);
```

NOTE：使用它来实现动画与 `setTimeout` 类似，需要每次每帧结束时再次调用。不可设置时间间隔（系统决定），时间间隔为16.67毫秒一帧。

常见动画

大多的复杂动画都是有下列的简单动画所组成的。

- 形变，改变元素的宽高
- 位移，改变元素相对位置
- 旋转
- 透明度
- 其他...

动画函数

下面的例子为以 px 为单位的动画代码

```
var animation = function(ele, attr, from, to) {
  var distance = Math.abs(to - from);
  var stepLength = distance/100;
  var sign = (to - from)/distance;
  var offset = 0;
  var step = function(){
    var tmpOffset = offset + stepLength;
    if (tmpOffset < distance) {
      ele.style[attr] = from + tmpOffset * sign + 'px';
      offset = tmpOffset;
    } else {
      ele.style[attr] = to + 'px';
      clearInterval(intervalID);
    }
  }
  ele.style[attr] = from + 'px';
  var intervalID = setInterval(step, 10);
}
```

表单操作

表单为页面的主要组成部分，其中包含许多的表单控件。用户通过控件提供数据并提交给服务器，服务器则做出相应的处理。而编写一个正常工作的表单需要三个部分：

1. 构建表单
2. 服务器处理（提供接受数据接口）
3. 配置表单

构建表单

```
<form>
  <p><label>姓名: <input></label></p>
  <p><label>电话: <input type="tel"></label></p>
  <p><label>邮箱: <input type="email"></label></p>
  <fieldset>
    <legend> 披萨大小 </legend>
    <label> <input type="radio" name="size"> 小 </label>
    <label> <input type="radio" name="size"> 中 </label>
    <label> <input type="radio" name="size"> 大 </label>
  </fieldset>
  <fieldset>
    <legend> 披萨配料 </legend>
    <label> <input type="checkbox"> 熏肉 </label>
    <label> <input type="checkbox"> 奶酪 </label>
    <label> <input type="checkbox"> 洋葱 </label>
    <label> <input type="checkbox"> 蘑菇 </label>
  </fieldset>
  <p><label>配送时间: <input type="time"
      min="11:00" max="21:00" step="900"></label></p>
  <p><button>提交订单</button></p>
</form>
```

```
<form>
  <p><label>姓名：<input></label></p>
  <p><label>电话：<input type="tel"></label></p>
  <p><label>邮箱：<input type="email"></label></p>
  <fieldset>
    <legend> 披萨大小 </legend>
    <label><input type="radio" name="size"> 小 </label>
    <label><input type="radio" name="size"> 中 </label>
    <label><input type="radio" name="size"> 大 </label>
  </fieldset>
  <fieldset>
    <legend> 披萨配料 </legend>
    <label><input type="checkbox"> 熏肉 </input></label>
    <label><input type="checkbox"> 奶酪 </input></label>
    <label><input type="checkbox"> 洋葱 </input></label>
    <label><input type="checkbox"> 蘑菇 </input></label>
  </fieldset>
  <p><label>配送时间：<input type="time" min="11:00" max="2100" st
ep="900"></label></p>
  <p><button>提交订单</button></p>
</form>
```

服务器处理

提供接口地址（例如，`https://pizza.example.com/order`，数据格式(`application/x-www-form-urlencoded`)，还是接受的参数信息(custname、custtel、custemail、size、topping、delivery)。

数据命名需在表单控件中注明。

配置表单

```
<form method="post"
      action="https://pizza.example.com/order"
      enctype="application/x-www-form-urlencoded">
  <p><label>姓名: <input name="custname"></label></p>
  <p><label>电话: <input type="tel" name="custtel"></label></p>
  <p><label>邮箱: <input type="email" name="custemail"></label></p>
  <fieldset>
    <legend> 披萨大小 </legend>
    <label> <input type="radio" name="size" value="small"> 小 </label>
    <label> <input type="radio" name="size" value="medium"> 中 </label>
    <label> <input type="radio" name="size" value="large"> 大 </label>
  </fieldset>
  <fieldset>
    <legend> 披萨配料 </legend>
    <label> <input type="checkbox" name="topping" value="bacon"> 熏肉 </label>
    <label> <input type="checkbox" name="topping" value="cheese"> 奶酪 </label>
    <label> <input type="checkbox" name="topping" value="onion"> 洋葱 </label>
    <label> <input type="checkbox" name="topping" value="mushroom"> 蘑菇 </label>
  </fieldset>
  <p><label>配送时间: <input type="time" name="delivery" min="11:00" max="21:00" step="900"></label></p>
  <p><button>提交订单</button></p>
</form>
```

```
<form action="https://pizza.example.com/order" method="post" enc
type="application/x-www-form-urlencoded">
  <p><label>姓名：<input name="custname"></label></p>
  <p><label>电话：<input type="tel" name="custtel"></label></p>
  <p><label>邮箱：<input type="email" name="custemail"></label></p
>
  <fieldset>
    <legend> 披萨大小 </legend>
    <label><input type="radio" name="size" value="small"> 小 </l
abel>
    <label><input type="radio" name="size" value="medium"> 中 </
label>
    <label><input type="radio" name="size" value="large"> 大 </l
abel>
  </fieldset>
  <fieldset>
    <legend> 披萨配料 </legend>
    <label><input type="checkbox"> 熏肉 </input></label>
    <label><input type="checkbox"> 奶酪 </input></label>
    <label><input type="checkbox"> 洋葱 </input></label>
    <label><input type="checkbox"> 蘑菇 </input></label>
  </fieldset>
  <p><label>配送时间：<input type="time" min="11:00" max="2100" st
ep="900"></label></p>
  <p><button>提交订单</button></p>
</form>
```

用户所有提交的信息需在提交服务器前对其进行验证从而提高用户体验。

NOTE：表单验证 使用 `require` 来强制用户填写相应的信息。

内容

元素

form 元素

form 元素为构建表单中最重要的元素。

```
<form novalidate name="pizza" target="abc" method="post" autocom
plete="off" accept-charset="utf-8" action="http://pizza.example.
com/order" enctype="application/x-www-form-urlencoded">
```

其对应的信息则可以视为

字段	值
noValidate	true
target	abc
method	post
acceptCharset	utf-8
action	http://pizza.example.com/order
enctype	application/x-www-form-urlencoded
name	pizza
autocomplete	off

NOTE：前六项为表单提交相关的信息。

属性

- `name` 属性：可以用于获取表单节点元素。

```
var pizzaForm = document.forms.pizza;
```

- `autocomplete` 属性：有两个值 `on` 与 `off`，在设置为 `on` 时，可以自动对输入框进行补全（之前提交过的输入值，下图左）。

NOTE：在已经设置 `autocomplete="off"` 时依然出现提示框，大多数情况为浏览器设置的自动补全（可以强制关闭，需要时请搜索对应的解决方案）。

- `elements` 属性：为一个动态节点集合（更具 DOM 的变化进行变化），其用于归结该表单的子孙表单控件（除图标按钮外 `<input type="image">`）：
    - button
    - fieldset
    - input
    - keygen
    - object
    - output
    - select
    - textarea

此外还有归属于该表单的空间（依旧图片按键除外）代码如下所示。

```
<form id="a">
</form>
<label><input name="null" form="a"></label>
```

- `length` 属性：等价于 `elements.length` 来用于描述表单内节点集合的个数。

选取表单空间元素

```
<form name="test">
  <input name="a">
  <input name="b">
</form>
```

选取 `name="a"` 的控件可以使用下面的方法：

```
testForm.elements[0];
testForm.elements['a'];

// 操作 Form 表单的属性
testForm[0];
testForm['a'];
```

- `form[name]` 通过名称作为索引时有如下特点：
  - 返回 `id` 或者 `name` 为指定名称的表单空间（图标按键除外）
  - 如果结果为空，则返回 `id` 为指定名称的 `img` 元素（入下面代码所示）
  - 如果有多个同名元素，则返回的元素为动态节点集合
  - 一旦用指定名称取过改元素，之后则不论该元素的 `id` 或者 `name` 如何变化，只有节点存在则均可使用原名称来继续获取改节点。

无指定名称索引范例

```
<form name="test">
  <img id="a" src="sample.png">
</form>
```

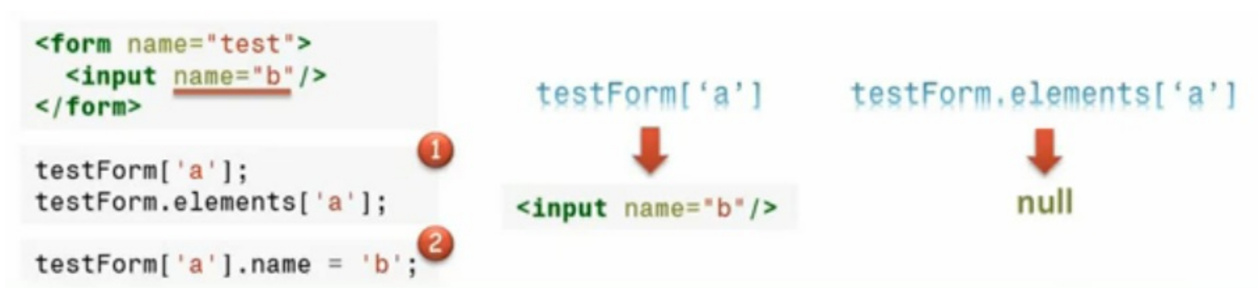
```
testForm['a']; // 取得的便是 id 为 a 的图片元素
```

更新名称，依然可以获取节点范例

```
<form name="test">
  <input name="a">
</form>
```

```
// 第一步
testForm['a'];
// 或者
testForm.elements['a'];

// 第二步
testForm['a'].name = 'b';
```



form 接口

`form` 元素也提供了一些接口便于对其进行操作 `reset()` `submit()` `checkValidity()` 。

可以重置（reset）的元素有下面的几种：

- input
- keygen
- output
- select
- textarea

当触发表单 `reset` 事件时可使用阻止该事件的默认行为来取消重置。而且元素重置时将不会再次触发元素上的 `change` 与 `input` 事件。



label 元素

```
<label for="textId" form="formId">
```

字段	值
htmlFor	textId
control	HTMLElement#textId
form	HTMLFormElement#formId

- `htmlFor` 属性：用于关联表单控件的激活行为（可使点击 `label` 与点击表单控件的行为一致），可关联的元素有下列（ `hidden` 除外）：
    - button
    - input
    - keygen
    - meter
    - output
    - progress
    - select
    - textarea

自定义文件提交控件样式



- `control` 属性：如果指定了 `for` 属性则指定该 `for` 属性对于 `id` 的可关联元素。如果没有指定 `for` 属性则为第一个可关联的子孙元素。

```
<label for="txtId">文字<input name="desc"></label>

<span id="txtId"> only for test content here </span>
```

① 指定了for属性

② for属性对应ID的元素span为非可关联元素

③ label.control → null

可关联的元素 （只读属性不可在程序中直接赋值修改）

- button
- fieldset
- input
- keygen
- label
- object
- output
- select
- textarea

— `form` 属性：修改关联元素所归属的表单则可以修改元素的 `form` 属性为带关联表单Id（元素中对于的 `for` 属性也应该做对应的修改）。//这里有一点小问题，更改form属性之后label并不能自动绑定到新表单对应的元素上

```
label.setAttribute('form', 'newFormId');
```

input 元素

```
<input type="text">
```

- `type` 属性：可用于控制控件的外观以及数据类型（默认为 `text` ），在不同的浏览器不同数据类型有不同的展示效果。

本地图片预览示例

所需技术点（HTMLInputElement属性）

- onchange
- accept
- multiple
- files

```
<input type="file" accept="image/*" multiple>
```

```
file.addEventListener(
  'change', function(event){
      var files = Array.prototype.slice.call(
        event.target.files, 0
      );
      files.forEach(function(item){
        files2dataurl(item,function(url){
          var image = new Image();
          parent.appendChild(image);
          image.src = url;
        });
      });
  }
);
function file2dataurl(file, callback) {
    if (!window.FileReader) {
        throw 'Browser not support File API !';
    }
    var reader = new FileReader();
    reader.readAsDataURL(file);
    reader.onload = function(event) {
        callback(event.target.result);
    };
}
```

NOTE： `accept` 所支持的格式有 `audio/*` `video/*` `image/*` 以及不带 `;` 的 MINE Type 类型和 `.` 开头的文件名后缀的文件。多个文件类型可以使用 `,` 分隔。

select 元素




```
<select name="course">
  <option>课程选择</option>
  <optgroup label="1. DOM基础">
    <option value="1.1">1.1 文档树</option>
    <option value="1.2">1.2 节点操作</option>
    <option value="1.3">1.3 元素遍历</option>
    <option value="1.4">1.4 样式操作</option>
    <option value="1.5">1.5 属性操作</option>
    <option value="1.6">1.6 表单操作</option>
  </optgroup>
  <optgroup label="2. 事件模型">
    <option value="2.1">2.1 事件类型</option>
    <option value="2.2">2.2 事件模型</option>
    <option value="2.3">2.3 事件应用</option>
  </optgroup>
</select>
```

指定选项列表中选择需要的选项。

主要的三个子标签 `select` 、 `optgroup` （用于选项分组）、 `option` 。

- `select` 具有的属性和方法如下：
    - name
    - value
    - multiple
    - options（动态节点集合）
    - selectedOptions（动态节点集合）
    - selectedIndex
    - add(element[, before])（无指定参照物则添加至最末端）
    - remove([index])
- `optgroup` 所具有的属性和方法：
    - disabled （分组选项不可选）
    - label（分组说明）
- `option` 所具有的属性和方法：
    - disabled
    - label（描述信息）
    - value（提交表单时的数据信息）
    - text（用户看到的文字）
    - index

  - selected
  - defaultSelected

选项操作

创建选项

```
document.createElement('option')
// 或者
new Option([text[, value[, defaultSelected[, selected]]]])
```

添加选项

```
var option = new Option('sample');
opt.insertAdjacentElement(option, '参照元素');
// 或者
select.add(option, '参照元素')
```

删除选项

```
opt.parentNode.removeChild(option);
// 或者使用它的索引将其删除
select.remove(2);
```

级联下列选择器

所需知识点:

- onchange
- remove
- add

```
<form name="course">
  <select name="chapter">
    <option>Select0</option>
  </select>
  <select name="section">
    <option>Select1</option>
  </select>
</form>
```

```
var chapters = {
  {text: 1, value: 1},
  {text: 2, value: 2}
};
var sections = {
  1: [{
    text:1.1, value: 1.1
  }, {
    text:1.2, value: 1.2
  }],
  2:[{
    text:2.1, value:2.1;
  }]
};

function fillSelect(select, list) {
  for(var i = select.length; i > 0; i--) {
    select.remove(i);
  }
  list.forEach(function(data){
    var option = new Option(data.text, data.value);
    select.add(option);
  })
}

fileSelect(chapterSelect, chapters);
chapterSelect.addEventListener(
  'change', function(event) {
    var value = event.target.value,
        list = sections[value] || [];
    fillSelect(sectionSelect, list);
  }
);
```

textarea 元素

`textarea` 具有的属性和方法如下：

- name

- value （用户输入信息）
- select() （全选当前输入的内容）
- selectionStart （选中的内容的起始位置，无选中时返回当前光标所在位置）
- selectionEnd （选中内容结束位置，无选中时返回光标位置）
- selectionDirection （选取方向 `forward` `backward` ）
- setSelectionRange(start, end[, direction]) （使用程序选中内容）
- setRangeText(replacement[, start, end, [mode]]) （设置内容范围）

selection

表示选择区域，对于 `input` 元素同样有效。



`selectionDirection` 主要是用于在使用 SHIFT 键与方向键组合选取时的选取方向。设置为 `forward` 时选取移动的方向为 `selectionEnd` 设置为 `backward` 时移动方向为 `selectionStart` 。

@输入提示示例



所需知识点：

- oninput
- selectionStart
- setRangeText

```
textarea.addEventListener(
  'input', function(event) {
    var target = event.target,
        cursor = target.selectionStart;
    if(target.value.charAt(cursor-1) === '@') {
      doShowAtList(functi=on(name){
        var end = cursor + name.length;
        target.setRangeText(
          name, cursor, end, 'end'
        );
      });
    }
  }
);
```

其他元素

- fieldset
- button
- keygen
- output
- progress
- meter

BROWSERS

Select up to five browsers and compare their test results in detail

	33 Internet Explorer 8	113 Internet Explorer 9	297 Internet Explorer 10	433 Chrome 24	404 Firefox 24
parsing rules	0	0	5	5	5
<!DOCTYPE html> triggers standards mode	Yes ✓	Yes ✓	Yes ✓	Yes ✓	Yes ✓
HTML5 tokenizer	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓
HTML5 tree building	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓
Parsing inline SVG	No ✗	Yes ✓	Yes ✓	Yes ✓	Yes ✓
Parsing inline MathML	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓
elements	2	12	12	24	21
Embedding custom non-visible data	No ✗	No ✗	No ✗	Yes ✓	Yes ✓
New or modified elements					
Section elements	No ✗	Yes ✓	Yes ✓	Yes ✓	Yes ✓
Grouping content elements	No ✗	Partial ○	Partial ○	Partial ○	Yes ✓
Text-level semantic elements	No ✗	Partial ○	Partial ○	Partial ○	Partial ○
Interactive elements	No ✗	No ✗	No ✗	Partial ○	No ✗
Global attributes or methods					
hidden attribute	No ✗	No ✗	No ✗	Yes ✓	Yes ✓
Dynamic markup insertion	Yes ✓	Yes ✓	Yes ✓	Yes ✓	Yes ✓
forms	3	3	34	61	39
Field types					
input type=text	Partial ○	Partial ○	Partial ○	Yes ✓	Yes ✓
input type=search	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓
input type=tel	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓
input type=url	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓
input type=email	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓
input type=date	No ✗	No ✗	No ✗	Yes ✓	No ✗
input type=month	No ✗	No ✗	No ✗	No ✗	No ✗
input type=week	No ✗	No ✗	No ✗	No ✗	No ✗
input type=time	No ✗	No ✗	No ✗	Yes ✓	No ✗
input type=datetime	No ✗	No ✗	No ✗	No ✗	No ✗
input type=datetime-local	No ✗	No ✗	No ✗	No ✗	No ✗
input type=number	No ✗	No ✗	Yes ✓	Yes ✓	No ✗
input type=range	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓
input type=color	No ✗	No ✗	No ✗	Yes ✓	No ✗
input type=checkbox	Partial ○	Yes ✓	Yes ✓	Yes ✓	Yes ✓
input type=image	No ✗	Yes ✓	Yes ✓	Yes ✓	Yes ✓
input type=file	No ✗	Partial ○	Yes ✓	Yes ✓	Yes ✓
textarea	Partial ○	Partial ○	Yes ✓	Yes ✓	Yes ✓
select	Partial ○	Partial ○	Yes ✓	Yes ✓	Yes ✓
fieldset	Partial ○	Partial ○	Partial ○	Yes ✓	Yes ✓
datalist	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓
keygen	No ✗	No ✗	No ✗	Yes ✓	No ✗
output	No ✗	No ✗	No ✗	Yes ✓	Yes ✓
progress	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓
meter	No ✗	No ✗	No ✗	Yes ✓	Yes ✓
Fields					
Field validation	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓
Association of controls and forms	No ✗	No ✗	Partial ○	Yes ✓	Partial ○
Other attributes	No ✗	No ✗	Partial ○	Yes ✓	Partial ○
CSS selectors	No ✗	No ✗	Partial ○	Yes ✓	Partial ○
Events	Partial ○	Partial ○	Partial ○	Yes ✓	Yes ✓
Forms					
Form validation	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓

	IE 8	IE 9	IE 10	Chrome 24	Firefox 24
microdata	0	0	0	0	0
Microdata	No ✗	No ✗	No ✗	No ✗	Yes ✓
location and orientation	0	15	15	18	20
Geolocation	No ✗	Yes ✓	Yes ✓	Yes ✓	Yes ✓
Device Orientation	No ✗	No ✗	No ✗	Yes ✓	Yes ✓
Device Motion	No ✗	No ✗	No ✗	No ✗	Yes ✓
output	0	0	0	10	10
Full screen support	No ✗	No ✗	No ✗	Prefixed ✓	Prefixed ✓
Web Notifications	No ✗	No ✗	No ✗	Yes ✓	Yes ✓
input	0	0	0	20	18
Access the webcam	No ✗	No ✗	No ✗	Prefixed ✓	Prefixed ✓
Gamepad control	No ✗	No ✗	No ✗	Prefixed ✓	No ✗
Pointer Events	No ✗	No ✗	No ✗	No ✗	No ✗
Pointer Lock support	No ✗	No ✗	No ✗	Prefixed ✓	Prefixed ✓
video	0	21	31	31	21
video element	No ✗	Yes ✓	Yes ✓	Yes ✓	Yes ✓
Subtitles	No ✗	No ✗	Yes ✓	Yes ✓	No ✗
Audio track selection	No ✗	No ✗	Yes ✓	No ✗	No ✗
Video track selection	No ✗	No ✗	No ✗	No ✗	No ✗
Poster images	No ✗	Yes ✓	Yes ✓	Yes ✓	Yes ✓
Codec detection	No ✗	Yes ✓	Yes ✓	Yes ✓	Buggy !
Advanced					
DRM support	No ✗	No ✗	No ✗	No ✗	No ✗
Media Source extensions	No ✗	No ✗	No ✗	Prefixed ✓	No ✗
Codecs					
MPEG-4 ASP support	No ✗	No ✗	No ✗	No ✗	No ✗
H.264 support	No ✗	Yes ✓	Yes ✓	Yes ✓	Yes ✓
Ogg Theora support	No ✗	No ✗	No ✗	Yes ✓	Yes ✓
WebM with VP8 support	No ✗	No ✗	No ✗	Yes ✓	Yes ✓
WebM with VP9 support	No ✗	No ✗	No ✗	No ✗	No ✗

	IE 8	IE 9	IE 10	Chrome 24	Firefox 24
audio	0	20	20	25	22
audio element	No ✗	Yes ✓	Yes ✓	Yes ✓	Yes ✓
Loop audio	No ✗	Yes ✓	Yes ✓	Yes ✓	Yes ✓
Preload in the background	No ✗	Yes ✓	Yes ✓	Yes ✓	Yes ✓
Advanced					
Web Audio API	No ✗	No ✗	No ✗	Prefixed ✓	No ✗
Speech Recognition	No ✗	No ✗	No ✗	No ✗	No ✗
Speech Synthesis	No ✗	No ✗	No ✗	No ✗	Yes ✓
Codecs					
PCM audio support	No ✗	No ✗	No ✗	Yes ✓	Yes ✓
AAC support	No ✗	Yes ✓	Yes ✓	Yes ✓	Yes ✓
MP3 support	No ✗	Yes ✓	Yes ✓	Yes ✓	Yes ✓
Ogg Vorbis support	No ✗	No ✗	No ✗	Yes ✓	Yes ✓
Ogg Opus support	No ✗	No ✗	No ✗	No ✗	Yes ✓
WebM with Vorbis support	No ✗	No ✗	No ✗	Yes ✓	Yes ✓
WebM with Opus support	No ✗	No ✗	No ✗	No ✗	No ✗
responsive images	0	0	0	0	0
picture element	No ✗	No ✗	No ✗	No ✗	No ✗
srcset attribute	No ✗	No ✗	No ✗	No ✗	No ✗
sizes attribute	No ✗	No ✗	No ✗	No ✗	No ✗
2d graphics	0	12	12	14	17
Canvas 2D graphics	No ✗	Yes ✓	Yes ✓	Yes ✓	Yes ✓
Drawing primitives					
Text support	No ✗	Yes ✓	Yes ✓	Yes ✓	Yes ✓
Path support	No ✗	No ✗	No ✗	No ✗	No ✗
Ellipse support	No ✗	No ✗	No ✗	No ✗	No ✗
Dashed line support	No ✗	No ✗	No ✗	Yes ✓	No ✗
System focus ring support	No ✗	No ✗	No ✗	No ✗	No ✗
Features					
Hit testing support	No ✗	No ✗	No ✗	No ✗	No ✗
Blending modes	No ✗	No ✗	No ✗	No ✗	Yes ✓
Image export formats					
PNG support	No ✗	Yes ✓	Yes ✓	Yes ✓	Yes ✓
JPEG support	No ✗	Yes ✓	Yes ✓	Yes ✓	Yes ✓
JPEG-XR support	No ✗	No ✗	No ✗	No ✗	No ✗
WebP support	No ✗	No ✗	No ✗	Yes ✓	No ✗
3d graphics	0	0	0	20	20
WebGL 3D graphics	No ✗	No ✗	No ✗	Prefixed ✓	Yes ✓
animation	0	0	5	5	5
window.requestAnimationFrame	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓

	IE 8	IE 9	IE 10	Chrome 24	Firefox 24
communication	0	0	27	33	33
Server-Sent Events	No ✗	No ✗	No ✗	Yes ✓	Yes ✓
Beacon	No ✗	No ✗	No ✗	No ✗	No ✗
XMLHttpRequest Level 2					
Upload files	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓
Response type support	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓
WebSocket					
Basic socket communication	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓
ArrayBuffer and Blob support	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓
peer to peer	0	0	0	15	15
WebRTC 1.0	No ✗	No ✗	No ✗	Prefixed ✓	Prefixed ✓
ObjectRTC API for WebRTC	No ✗	No ✗	No ✗	No ✗	No ✗
Data channel	No ✗	No ✗	No ✗	Prefixed ✓	Prefixed ✓
user interaction	9	9	12	13	19
Drag and drop					
Attributes	No ✗	No ✗	Partial ○	Yes ✓	Partial ○
Events	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓
HTML editing					
Editing elements	Yes ✓	Yes ✓	Yes ✓	Yes ✓	Yes ✓
Editing documents	Yes ✓	Yes ✓	Yes ✓	Yes ✓	Yes ✓
CSS selectors	No ✗	No ✗	No ✗	No ✗	Yes ✓
APIs	Yes ✓	Yes ✓	Yes ✓	Yes ✓	Yes ✓
Clipboard					
Clipboard API and events	No ✗	No ✗	No ✗	No ✗	Yes ✓
Spellcheck					
spellcheck attribute	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓
performance	0	0	20	25	21
Native binary data	No ✗	No ✗	Partial ○	Yes ✓	Yes ✓
Workers					
Web Workers	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓
Shared Workers	No ✗	No ✗	No ✗	Yes ✓	No ✗
security	5	5	20	28	23
Web Cryptography API	No ✗	No ✗	No ✗	No ✗	No ✗
Content Security Policy 1.0	No ✗	No ✗	No ✗	Yes ✓	Yes ✓
Content Security Policy 1.1	No ✗	No ✗	No ✗	No ✗	No ✗
Cross-Origin Resource Sharing	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓
Cross-document messaging	Yes ✓	Yes ✓	Yes ✓	Yes ✓	Yes ✓
Iframes					
Sandboxed iframe	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓
Seamless iframe	No ✗	No ✗	No ✗	No ✗	No ✗
iframe with inline contents	No ✗	No ✗	No ✗	Yes ✓	No ✗
history and navigation	0	0	10	10	10
Session history	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓

	IE 8	IE 9	IE 10	Chrome 24	Firefox 24
web applications	1	1	11	13	15
Offline resources					
Application Cache	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓
Service Workers	No ✗	No ✗	No ✗	No ✗	No ✗
Content and Scheme handlers					
Custom scheme handlers	No ✗	No ✗	No ✗	Yes ✓	Yes ✓
Custom content handlers	No ✗	No ✗	No ✗	No ✗	Yes ✓
Custom search providers	Yes ✓	Yes ✓	Yes ✓	Yes ✓	Yes ✓
storage	10	10	35	33	35
Key-value storage					
Session Storage	Yes ✓	Yes ✓	Yes ✓	Yes ✓	Yes ✓
Local Storage	Yes ✓	Yes ✓	Yes ✓	Yes ✓	Yes ✓
Database storage					
IndexedDB	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓
Objectstore Blob support	No ✗	No ✗	Yes ✓	No ✗	Yes ✓
Objectstore ArrayBuffer support	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓
Web SQL Database	No ✗	No ✗	No ✗	Yes ✓	No ✗
files	0	0	15	15	15
Reading files					
Basic support for reading files	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓
Create a Blob from a file	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓
Create a Data URL from a Blob	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓
Create an ArrayBuffer from a Blob	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓
Create a Blob URL from a Blob	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓
Accessing the file system					
FileSystem API	No ✗	No ✗	No ✗	No ✗	No ✗
File API: Directories and System	No ✗	No ✗	No ✗	Prefixed ✓	No ✗
streams	0	0	0	0	0
Readable streams	No ✗	No ✗	No ✗	No ✗	No ✗
Writable streams	No ✗	No ✗	No ✗	No ✗	No ✗

	IE 8	IE 9	IE 10	Chrome 24	Firefox 24
web components	0	0	0	0	2
Custom elements	No ✗	No ✗	No ✗	No ✗	No ✗
Shadow DOM	No ✗	No ✗	No ✗	No ✗	No ✗
HTML templates	No ✗	No ✗	No ✗	No ✗	Yes ✓
HTML imports	No ✗	No ✗	No ✗	No ✗	No ✗
other	3	5	13	15	18
Styling					
Scoped style element	No ✗	No ✗	No ✗	No ✗	Yes ✓
Scripts					
Asynchronous script execution	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓
Runtime script error reporting	Yes ✓	Yes ✓	Yes ✓	Yes ✓	Yes ✓
Script execution events	No ✗	No ✗	No ✗	No ✗	Yes ✓
Base64 encoding and decoding	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓
JSON encoding and decoding	Yes ✓	Yes ✓	Yes ✓	Yes ✓	Yes ✓
URL API	No ✗	No ✗	Yes ✓	Yes ✓	Yes ✓
Mutation Observer	No ✗	No ✗	No ✗	Prefixed ✓	Yes ✓
Promises	No ✗	No ✗	No ✗	No ✗	No ✗
Other					
Page Visibility	No ✗	No ✗	Yes ✓	Prefixed ✓	Yes ✓
Text selection	No ✗	Yes ✓	Yes ✓	Yes ✓	Yes ✓
Scroll into view	Yes ✓	Yes ✓	Yes ✓	Yes ✓	Yes ✓

验证

可以被验证的元素如下所示：】

- button
- input
- select
- textarea

以下情况不可以做验证

- input 元素在类型是 hidden, reset, button 时
- button 元素在类型为 reset, button 时
- input 与 textarea 当属性为 readonly 时
- 当元素为 datalist 的子孙节点时
- 当元素被禁用时 disabled 的状态

属性

验证涉及到以下的以下属性，在每一个可以验证的元素上均可以调用对于的属性或通过接口进行操作：

- willValidate （表明此元素在表单提交时是否会被验证）
- checkValidity() （用于验证元素，返回 true 当验证通过，或者触发 invalid 事件）
- validity （存储验证结果）
- validationMessage （显示验证异常信息）
- setCustomValidity(message) （自定义验证错误信息）

名称	描述
valueMissing	设置了required没有value
typeMismatch	value与type不符，如email，url
patternMismatch	value与pattern不匹配
tooLong	value长度超过maxlength指定的长度
tooShort	value长度小于minlength指定的长度
rangeUnderflow	value值小于min指定的值
rangeOverflow	value值大于max指定的值
stepMismatch	value值不符合step指定的值
badInput	输入不完整
customError	使用setCustomValidity设置了自定义错误
valid	符合验证条件

自定义异常范例

涉及到的知识点：

- oninvalid
- setCustomValidity

```
<form action="./api" method="post">
  <label>Name: <input name="username" required></label>
  <button>submit</button>
</form>
```

```
input.addEventListener(
  'invalid', function(event){
    var target = event.target;
    if (target.validity.valueMissing) {
      target.setCustomValidity('Name is missing');
    }
  }
)
```

禁止验证范例

使用 `form` 中 `novalidate` 属性来禁止表单提交的验证。

```
<form action="./api" method="post" novalidate>
  <label>Mobile: <input name="mobile" type="number"></label>
  <button>submit</button>
</form>
```

提交

隐式提交

在操作过程中通过控件的操作来提交表单（敲击回车来提交表单），其需要满足以下的条件：

- 表单有非禁用的提交按键
- 没有提交按键时，不超过一个类型为 `text` `search` `url` `email` `password` `date` `time` `number` 的 `input` 元素

提交过程细节

提交过程分为两个阶段，第一个阶段是更具表单 enctype 指定的值构建要提交的数据，第二个阶段是使用指定的方法（method）发送数据到 `action` 指定的目标。

构建提交数据，从可提交元素中提取数据组成指定数据结构过程（可提交元素有 `button` `input` `keygen` `object` `select` `textarea` ）

编码方式（enctype）所支持的形式：

- application/x-www-form-urlencoded （默认，数据格式为 `&` 分隔的键值对）
- multipart/form-data （IFC 2388 字节流形式，例如文件上传所使用的数据编码形式）
- text/plain （回车换行符分隔的键值对）

```
<form action="./api"
      method="post"
      enctype="application/x-www-form-urlencoded"
>
  <inp
  <inp
  <inp
  <but
</form
```

a b c submit

```
POST http://127.0.0.1:8060/training/MOOC/DOM%e7
Host: 127.0.0.1:8060
Connection: keep-alive
Content-Length: 11
Pragma: no-cache
Cache-Control: no-cache
Accept: text/html,application/xhtml+xml,applicat
Origin: http://127.0.0.1:8060
User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64)
Content-Type: application/x-www-form-urlencoded
Referer: http://127.0.0.1:8060/training/MOOC/DOM
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.8

a=a&b=b&c=c
```

```
<form action="./api"
      method="post"
      enctype="multipart/form-data"
>
  <inp
  <inp
  <inp
  <but
</form
```

a b c submit

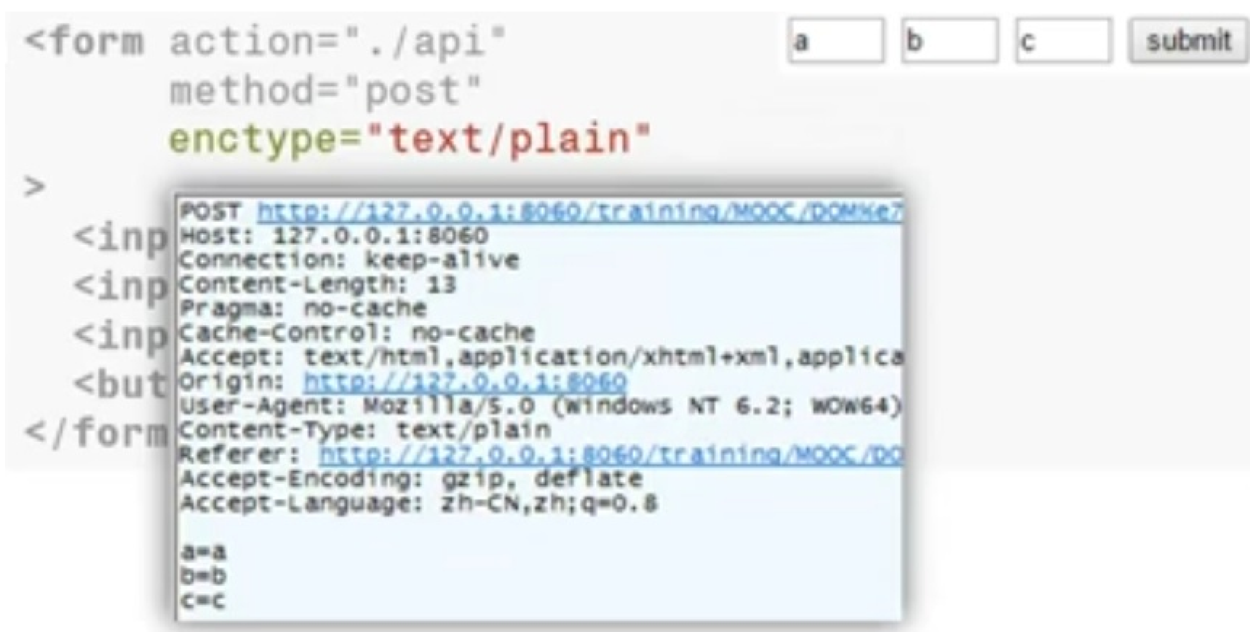
```
POST http://127.0.0.1:8060/training/MOOC/DOM%e7%...
Host: 127.0.0.1:8060
Connection: keep-alive
Content-Length: 311
Pragma: no-cache
Cache-Control: no-cache
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Origin: http://127.0.0.1:8060
User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Content-Type: multipart/form-data; boundary=----WebKitFormBoundarygVYNOTINGM8eXReQ
Referer: http://127.0.0.1:8060/training/MOOC/DOM%e7%...
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.8

------WebKitFormBoundarygVYNOTINGM8eXReQ
Content-Disposition: form-data; name="a"

a
------WebKitFormBoundarygVYNOTINGM8eXReQ
Content-Disposition: form-data; name="b"

b
------WebKitFormBoundarygVYNOTINGM8eXReQ
Content-Disposition: form-data; name="c"

c
------WebKitFormBoundarygVYNOTINGM8eXReQ--
```



特殊案例一

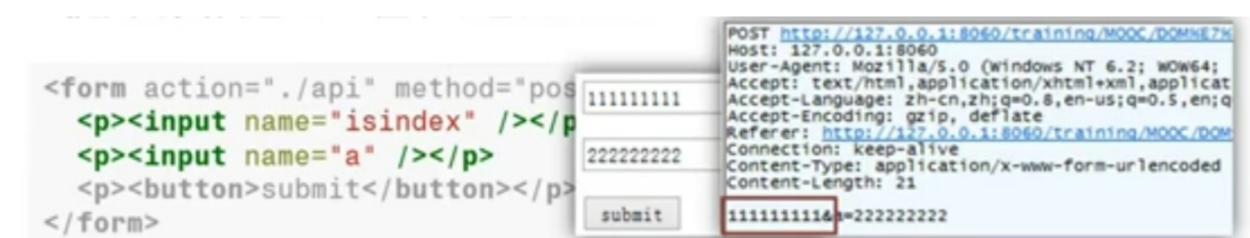
当一个表单元素 `name="isindex"` 并且 `type="text"` 而且满足如下要求时：

- 编码格式为 application/x-www-form-urlencoded
- 作为表单的第一个元素

则提交时只发送 value 值，不包含 name。



```
<form action="./api" method="post">
  <input name="isindex">
  <input name="a">
  <button>submit</button>
</form>
```

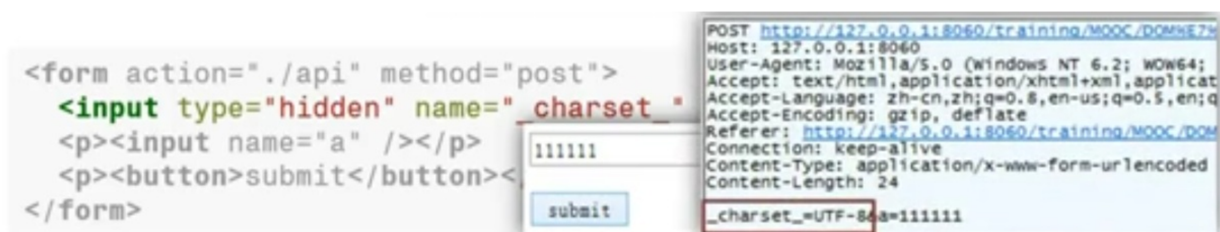


特殊案例二

当 `name="_charset_"` 并且类型为 `hidden` 时，而且满足如下要求时：

- 没有设置 `value` 值

则提交时 `value` 自动使用当前提交的字符集填充。



submit 接口

`form.submit()` 可以通过调用接口 `submit()` 直接提交表单，在提交表单时均会触发一个 `onsubmit` 表单提交事件，在这个事件中 women 可以做下面的事件：

- 提交前的数据验证
- 阻止事件的默认行为来取消表单的提交（例如当验证失败时）

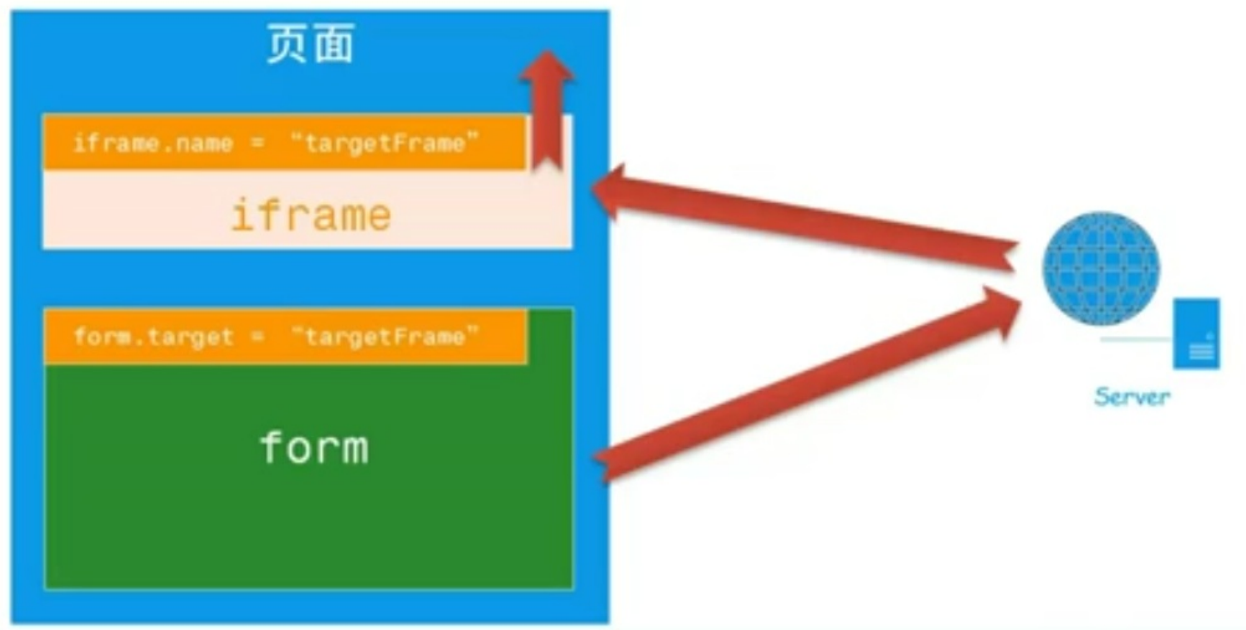
```
form.addEventListener(
  'submit', function(event) {
    var notValid = false;
    var elements = event.target.elements;

    // 自定义验证

    if (notValid) {
      // 取消提交
      event.preventDefault();
    }
  }
)
```

无刷新表单提交范例

常用的方式是通过 AJAX 进行实现，这里我们使用 iframe 来做中介代理实现。



所需知识点：

- form
- target
- iframe

```
<iframe name="targetFrame" class="f-hidden" style="display:none"
id="result">

<form action="./api" method="post" target="targetFrame">
  <input name="isindex">
  <input name="a">
  <button>submit</button>
</form>
```

```
var frame = document.getElementById('result');
frame.addEventListener(
  'load', function(event) {
    try {
      var result = JSON.parse(
        frame.contentWindow.document.body.textContent
      );

      // 还原登陆按钮状态
      disabledSubmit(false);

      // 识别登陆结果
      if (result.code === 200) {
        showMessage('j-suc', 'success');
        form.reset();
      }
    } catch(ex) {
      // 忽略操作
    }
  }
)
```

表单应用

首先需要知道服务器端登陆接口的相关信息，如下所示：

描述	数据信息
请求地址	/api/login
请求参数	`telephone` 手机号码; `password` 密码 MD5 加密
返回结果	`code` 请求状态; `result` 请求数据结果

```html
<form action="/api/login" class="m-form" name="loginForm" target="result" autocomplete="off">
  <legend>手机号码登录</legend>
  <fieldset>
    <div class="msg" id="message"></div>
    <div>
      <label for="telephone">手机号：</label>
      <input id="telephone" name="telephone" class="u-txt"
             type="tel" maxlength="11" required
             pattern="^0?(13[0-9]|15[012356789]|18[0236789]|14[57])[0-9]{8}$"><br/>
      <span class="tip">11位数字手机号码</span>
    </div>
    <div>
      <label for="password">密  码：</label>
      <input id="password" name="password" type="password" class="u-txt"><br/>
      <span class="tip">至少6位，同时包含数字和字母</span>
    </div>
    <div><button name="loginBtn">登  录</button></div>
  </fieldset>
</form>
```

```javascript
var form = document.forms.loginForm;

var message = document.getElementById('message');

// 通用逻辑封装
function showMessage(class, message) {
  if(!class) {
    message.innerHTML = "";
    message.classList.remove('j-suc');
    message.classList.remove('j-err');
  } else {
    message.innerHTML = message;
    message.classList.add(class);
  }
}

function invalidInput (node, message) {
  showMessage('j-err', message);
  node.classList.add('j-err');
  node.focus();
}

function clearInvalid(node){
  showMessage();
  node.classList.remove('j-err');
}

function disabledSubmit(disabled) {
  form.loginBtn.disabled = !!disabled;
```

```
  var method = !disabled ? 'remove' : 'add';
  form.loginBtn.classList[method]('j-disabled');
}

// 验证手机号码（系统自带方法）
form.telephone.addEventListener(
  'invalid', function(event) {
    event.preventDefault();
    invalidInput(form.telephone, 'invalid mobile number');
  }
);

// 验证密码
form.addEventListener(
  'submit', function(event) {
    var input = form.password;
    var password = input.value;
    errorMessage = '';
    if (password.length < 6) {
      errorMessage = 'password less than 6 char';
    } else if (!/\d./test(password) || !/[a-z]/i.test(password))
 {
      errorMessage = 'password must contains number and letter'
    }

    if (!!errorMessage) {
      event.preventDefault();
      invalidInput(input, errorMessage);
      return;
    }
    // 提交表单代码
    // ...
  }
);

// 提交表单
form.addEventListener(
  'submit', function(event){
    input.value = md5(password);
    disabledSubmit(true);
```

```
  }
);

// 状态恢复
form.addEventListener(
  'focus', function(event) {
    // 错误还原
    clearInvalid(event.target);
    // 还原登陆按钮状态
    disabledSubmit(false);
  }
)
```

列表操作



列表的常用形式有图片形式与信息形式的，常见的有如下的操作：

- 显示列表
- 选择列表项
- 新增列表项
- 删除列表项
- 更新列表项

范例代码

数据结构



字段	说明
id	歌曲标识
name	歌曲名称
duration	歌曲时长
album	专辑信息
artist	歌手信息

字段	说明
id	歌手标识
name	歌手名称

字段	说明
id	专辑标识
name	专辑名称

```
[
  {
    "id": 22341234,
    "name": "Good Song",
    "album": {
      "id": 213512,
      "name": "Good Album"
    },
    "artist":{
      "id": 1234512,
      "name": "Evil Artist"
    }
  }
]
```

显示列表

列表容器

```
<div class="m-plylist m-plylist-sort s-bfc5" id="parent">
  <div class="head f-cb">
    <div class="fix">
      <div class="th col f-pr"></div>
      <div class="th col o-love">
        <span class="ico u-icn4 u-icn4-love"></span>
      </div>
    </div>
    <div class="flow f-cb">
      <div class="th col">音乐标题</div>
      <div class="th col">歌手</div>
      <div class="th col">专辑</div>
      <div class="th col">时长</div>
    </div>
  </div>

</div>
```

列表模板 分离数据和视图

```
<ul>
  {list tracks as track}
  <li class="itm j-item">
    <span class="ico u-icn4 u-icn4-love"></span>
    <div class="flow f-cb">
      <div class="td col title">
        <a href="/track/${track.id}/" class="tit s-bfc8">${track.name}</a>
      </div>
      <div class="td col ellipsis">
        <a href="/artist/${track.artist.id}/" class="s-bfc8">${track.artist.name}</a>
      </div>
      <div class="td col ellipsis">
        <a href="/album/${track.album.id}/" class="s-bfc4">${track.album.name}</a>
      </div>
      <div class="td col">${track.duration|dur2str}</div>
    </div>
  </li>
  {/list}
</ul>
```

绘制列表

准备数据并整合模板与数据

```
function render(parent,list){
    var ext = {
        dur2str:function(duration){
            duration = duration/1000;
            var m = Math.floor(duration/60),
                s = Math.floor(duration%60);
            return (m<10?'0':'')+m+':'+(s<10?'0':'')+s;
        }
    };
    var html = Trimpath.merge(
        tplContent,{tracks:list},ext
    );
    parent.insertAdjacentHTML('beforeEnd',html);
}
```

通过 *AJAX* 获取数据

```
var xhr = new XMLHttpRequest();
xhr.open('GET','/api/track',true);
xhr.onload = function(){
    render(
        document.getElementById('parent'),
        JSON.parse(xhr.responseText)
    );
};
xhr.send(null);
```

列表单选操作

```
parent.addEventListener(
    'mousedown',function(event){
        var target = getTarget(event);
        if (!!target&&!isSelected(target)&&
            !event.ctrlKey&&!event.shiftKey){
            clearSelection();
            appendToSelection(target);
        }
    }
);
```

列表多选操作（*Control* 与 *Shift* 操作）

```
parent.addEventListener(
  'mouseup', function(event) {
    var target = getTarget(event),
        selected = isSelected(target);
    // right click
    if (event.button == 2&&selected) {
      return;
    }
    // with control click
    if (event.ctrlKey) {
      !selected?appendToSelection(target):removeFromSelection(ta
rget);
    }
    // with shift key
    if (event.shiftKey) {
      var list = Array.prototype.slice.call(
        parent.getElementsByTagName('li'), 0
      );
      if (!last) {
        last = getLastElection() || target;
      }
      selectWithRangeFromTo(list, last, target);
    } else {
      last = null;
    }
  }
);
```

右键菜单

`contextmenu` 事件为右键菜单弹出事件。

```
parent.addEventListener(
    'contextmenu',function(event){
        var target = getTarget(event);
        if (!!target){
            event.preventDefault();
            showContextMenu(
                selection,
                event.pageX,
                event.pageY
            );
        }
    }
);
```

```
function showContextMenu(selection,left,top){
    // build menu items
    var actions = [
        {text:'删除歌曲',value:'delete'}
    ];
    if (selection.length<=1){
        actions.push(
            {text:'插入歌曲',value:'insert'},
            {text:'编辑歌曲',value:'update'}
        );
    }
    // show menu
    var menu = getMenu(
        TrimPath.merge(
            tplMenu,{actions:actions}
        )
    );
    menu.style.top = top+20+'px';
    menu.style.left = left+10+'px';
    document.body.appendChild(menu);
}
```

增加列表

```
function insertTrack(){
    showTrackAddForm(function(track){
        selection[0].insertAdjacentElement(
            'beforeBegin', getTrackItem(track)
        );
    });
}
```

```
var getTrackItem = function(track){
    var div = document.createElement('div');
    render(div,[track]);
    return div.getElementsByTagName('li')[0];
};
```

删除列表

```
function removeTrack(){
    selection.forEach(function(node){
        node.parentNode.removeChild(node);
    })
    selection = [];
}
```

更新列表

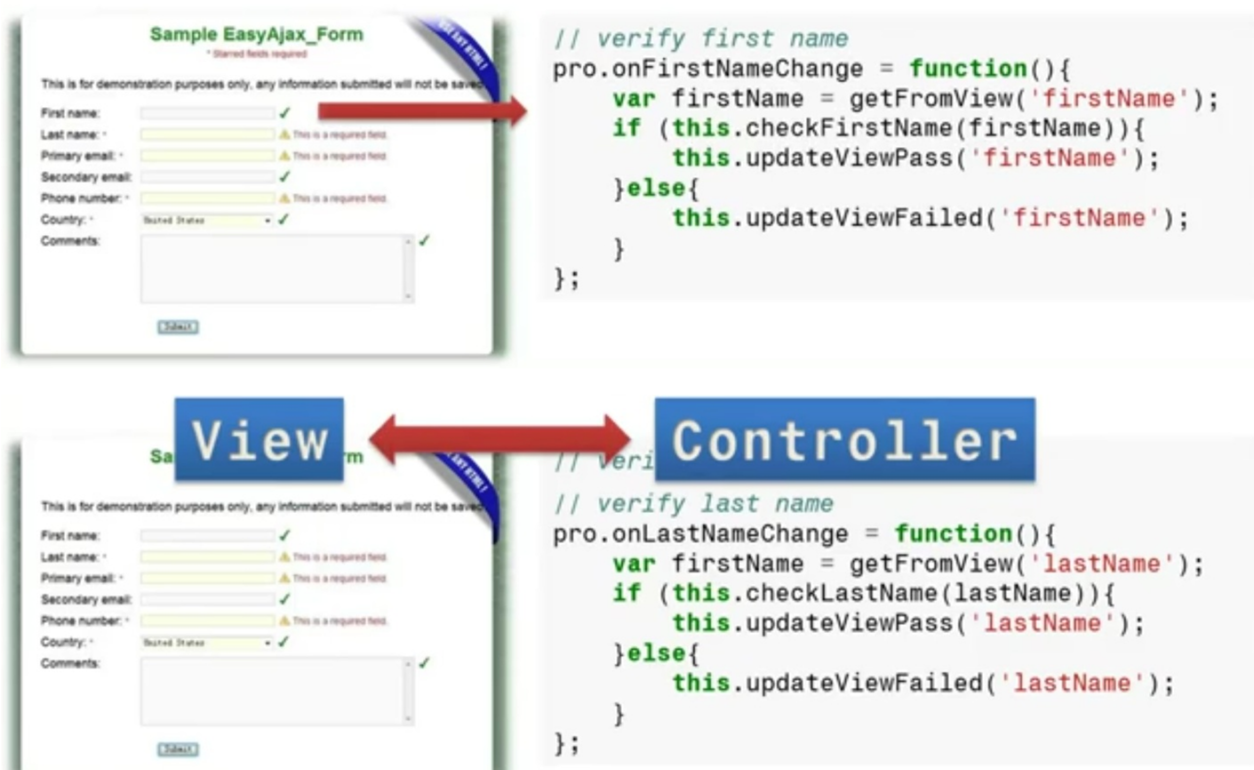
```
function updateTrack(){
    var node = selection[0];
    showTrackUpdateForm(
        node,function(track){
            var list = node.getElementsByTagName('a');
            list[0].textContent = track.name;
            list[1].textContent = track.album.name;
            list[2].textContent = track.artist.name;
        }
    );
}
```

更新状态

```
parent.addEventListener(
    'mousedown',function(event){
        // love all track
        var target = getTarget(event,'j-lvbtn');
        if (!!target){
            var loved = toggleLove(target);
            loved ? loveAll() : unloveAll();
            return;
        }
        // love one track
        var target = getTarget(event,'j-love');
        if (!!target){
            toggleLove(target);
            syncLoveAllState();
        }
    }
);
```

编程方式

面向视图 的操作方式，即为针对视图的直接编程（对 DOM 树进行直接的操作）。



这样的方式代理了许多的弊端，例如无法进行完全的自动化测试以及极高的视图层和控制层耦合的紧密性。

面向数据 的操作方式，视图则被抽象为若干的数据以及状态（后续所有的操作都会更加数据模型而操作），从而做到视图模型层完全自动化的测试。

```
// view model
this.data = {
    firstName : 'a',
    lastName : 'bcdef',
    status:{
        firstName : STATUS_NOT_CHECK,
        lastName : STATUS_NOT_CHECK
    }
};
```

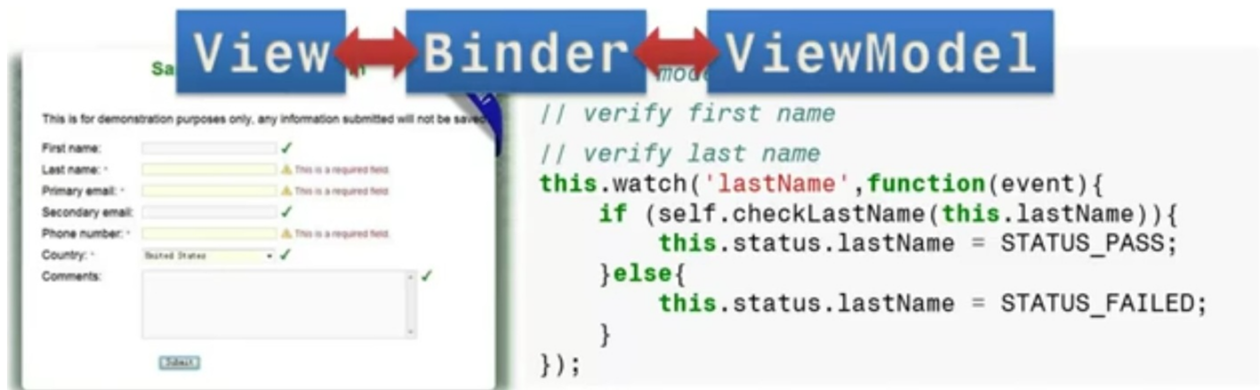
```
// view model
// verify first name
this.watch('firstName',function(event){
    if (self.checkFirstName(this.firstName)){
        this.status.firstName = STATUS_PASS;
    }else{
        this.status.firstName = STATUS_FAILED;
    }
});
```

```
// view model
// verify first name
// verify last name
this.watch('lastName',function(event){
    if (self.checkLastName(this.lastName)){
        this.status.lastName = STATUS_PASS;
    }else{
        this.status.lastName = STATUS_FAILED;
    }
});
```

View

ViewModel

```
// view model
// verify first name
// verify last name
this.watch('lastName',function(event){
    if (self.checkLastName(this.lastName)){
        this.status.lastName = STATUS_PASS;
    }else{
        this.status.lastName = STATUS_FAILED;
    }
});
```


View
Binder
ViewModel
This is for demonstration purposes only, any information submitted will not be save
First name:
Last name: *
Primary email: *
Secondary email:
Phone number: *
Country: *
Comments:
// verify first name
// verify last name
this.watch('lastName',function(event){
if (self.checkLastName(this.lastName)){
this.status.lastName = STATUS_PASS;
}else{
this.status.lastName = STATUS_FAILED;
}
});

页面架构

页面架构从实际需求出发，提供多种需求的多种解决方案和优缺点， 各种页面优化方案以及如何为团队或产品制定规范等， 帮助你完成产品的页面架构，进一步提升前端页面技术。

本章包括布局解决方案、响应式、页面优化、规范与模块化等。

CSS Reset

所有的 HTML 标签在没有设置样式时均被浏览器默认的样式列表所装饰 （不同浏览器默认样式有所不同）。**CSS** 的样式重置就是清楚浏览器的默认样式 ，可以理解为对于全局的样式定义。 对于开发者来言，如不重置每一个浏览器特定的默认样式，则会在开发造成诸多的不便。

> 在前端开发过程中做加法，远远比做减法简单。将所有浏览器的默认样式统一， 可以使它们有一个相同起点。
>
> — Li Xinyang 资深前端工程师

NOTE：一份 CSS Reset 文件并不一定适用于所有场景，需要更具需求做出变通（需符合产品需求为主）。

NOTE+：浏览器对于控件的样式和功能的特性支持也可以重置 （例如日历，清楚输入框内容按键等）。

NOTE++：Reset 文件需比页面其他样式文件的引入顺序优先（优先级需最高）。

样式重置前后对比

样式重置前

article
nav

h1

h2

h3

h4

h5

h6

address

p

pre

blockquote

1. ol li

- ul li

dl dt
dl dd

table caption
table thead tr th
table tbody tr td
table tfoot tr td
form
fieldset legend

label input text input button button button select option textarea ~~del~~ ins a *em* **strong** *i* **b** u s small abbr "q" *cite* *dfn* sub sup time code kbd samp *var* mark

样式重置后

article
nav
h1
h2
h3
h4
h5
h6
address
p
pre
blockquote
ol li
ul li
dl dt
dl dd
table caption
table thead tr th
table tbody tr td
table tfoot tr td
form
fieldset legend

label input text input button button button select option textarea del ins a em strong i b u s small abbr q cite dfn sub sup time code kbd samp var mark

布局解决方案

了解 CSS 中属性的值及其特性，透彻分析问题和需求才可以选择和设计最适合的布局解决方案。

居中布局

水平居中



子元素于父元素水平居中且其（子元素与父元素）宽度均可变。

inline-block + text-align

```
<div class="parent">
  <div class="child">Demo</div>
</div>

<style>
  .child {
    display: inline-block;
  }
  .parent {
    text-align: center;
  }
</style>
```

优点

- 兼容性佳（甚至可以兼容 IE 6 和 IE 7）

table + margin

```
<div class="parent">
  <div class="child">Demo</div>
</div>

<style>
  .child {
    display: table;
    margin: 0 auto;
  }
</style>
```

NOTE: `display: table` 在表现上类似 `block` 元素，但是宽度为内容宽。

优点

- 无需设置父元素样式 （支持 IE 8 及其以上版本）

NOTE：兼容 IE 8 一下版本需要调整为 `<table>` 的结果

absolute + transform

```
<div class="parent">
  <div class="child">Demo</div>
</div>

<style>
  .parent {
    position: relative;
  }
  .child {
    position: absolute;
    left: 50%;
    transform: translateX(-50%);
  }
</style>
```

优点

- 绝对定位脱离文档流，不会对后续元素的布局造成影响。

缺点

- `transform` 为 CSS3 属性，有兼容性问题

flex + justify-content

```
<div class="parent">
  <div class="child">Demo</div>
</div>

<style>
  .parent {
    display: flex;
    justify-content: center;
  }

  /* 或者下面的方法，可以达到一样的效果 */

  .parent {
    display: flex;
  }
  .child {
    margin: 0 auto;
  }
</style>
```

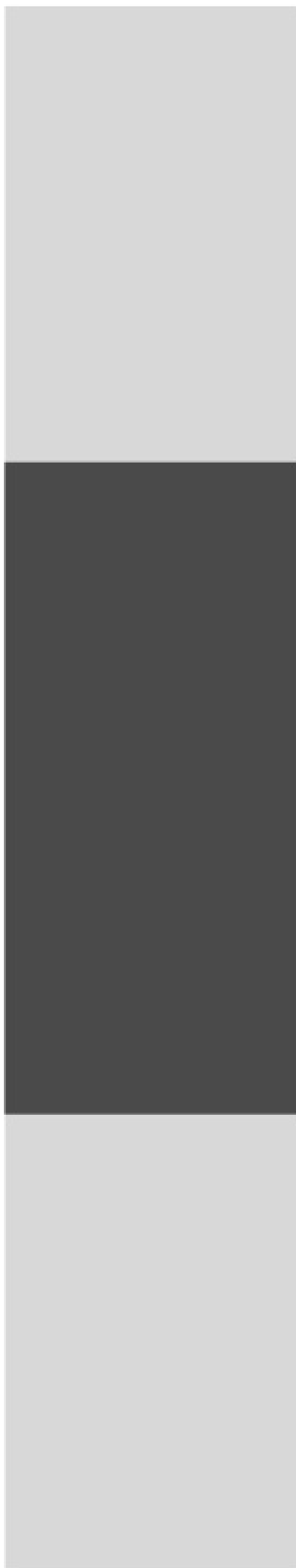
优点

- 只需设置父节点属性，无需设置子元素

缺点

- 有兼容性问题

垂直居中



子元素于父元素垂直居中且其（子元素与父元素）高度均可变。

table-cell + vertical-align

```
<div class="parent">
  <div class="child">Demo</div>
</div>

<style>
  .parent {
    display: table-cell;
    vertical-align: middle;
  }
</style>
```

优点

- 兼容性好（支持 IE 8，以下版本需要调整页面结构至 `table`）

absolute + transform

```
<div class="parent">
  <div class="child">Demo</div>
</div>

<style>
  .parent {
    position: relative;
  }
  .child {
    position: absolute;
    top: 50%;
    transform: translateY(-50%);
  }
</style>
```

优点

- 绝对定位脱离文档流，不会对后续元素的布局造成影响。但如果绝对定位元素是唯一的元素则父元素也会失去高度。

缺点

- `transform` 为 CSS3 属性，有兼容性问题

flex + align-items

```
<div class="parent">
  <div class="child">Demo</div>
</div>

<style>
  .parent {
    display: flex;
    align-items: center;
  }
</style>
```

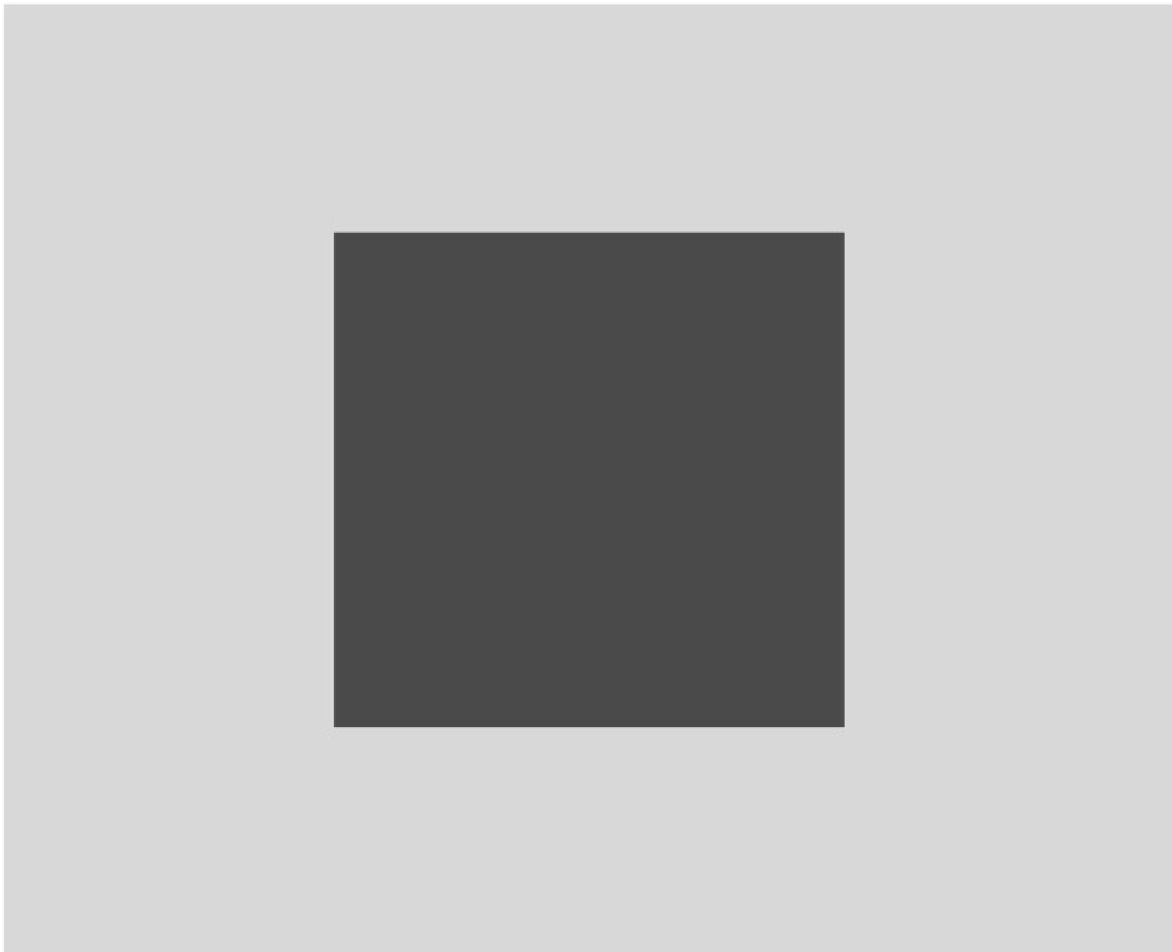
优点

- 只需设置父节点属性，无需设置子元素

缺点

- 有兼容性问题

水平与垂直居中



子元素于父元素垂直及水平居中且其（子元素与父元素）高度宽度均可变。

inline-block + text-align + table-cell + vertical-align

```
<div class="parent">
  <div class="child">Demo</div>
</div>

<style>
  .parent {
    text-align: center;
    display: table-cell;
    vertical-align: middle;
  }
  .child {
    display: inline-block;
  }
</style>
```

优点

- 兼容性好

absolute + transform

```
<div class="parent">
  <div class="child">Demo</div>
</div>

<style>
  .parent {
    position: relative;
  }
  .child {
    position: absolute;
    left: 50%;
    top: 50%;
    transform: translate(-50%, -50%);
  }
</style>
```

优点

- 绝对定位脱离文档流，不会对后续元素的布局造成影响。

缺点

- `transform` 为 CSS3 属性，有兼容性问题

flex + justify-content + align-items

```
<div class="parent">
  <div class="child">Demo</div>
</div>

<style>
  .parent {
    display: flex;
    justify-content: center;
    align-items: center;
  }
</style>
```

优点

- 只需设置父节点属性，无需设置子元素

缺点

- 有兼容性问题

多列布局

多列布局在网页中非常常见（例如两列布局），多列布局可以是两列定宽，一列自适应，或者多列不定宽一列自适应还有等分布局等。

一列定宽，一列自适应



float + margin

```
<div class="parent">
  <div class="left">
    <p>left</p>
  </div>
  <div class="right">
    <p>right</p>
    <p>right</p>
  </div>
</div>

<style>
  .left {
    float: left;
    width: 100px;
  }
  .right {
    margin-left: 100px
    /*间距可再加入 margin-left */
  }
</style>
```

NOTE：IE 6 中会有3像素的 BUG，解决方法可以在 `.left` 加入 `margin-left:-3px` 。

float + margin + (fix) 改造版

```
<div class="parent">
  <div class="left">
    <p>left</p>
  </div>
  <div class="right-fix">
    <div class="right">
      <p>right</p>
      <p>right</p>
    </div>
  </div>
</div>

<style>
  .left {
    float: left;
    width: 100px;
  }
  .right-fix {
    float: right;
    width: 100%;
    margin-left: -100px;
  }
  .right {
    margin-left: 100px
    /*间距可再加入 margin-left */
  }
</style>
```

NOTE：此方法不会存在 IE 6 中3像素的 BUG，但 `.left` 不可选择， 需要设置 `.left {position: relative}` 来提高层级。 此方法可以适用于多版本浏览器（包括 IE6）。缺点是多余的 HTML 文本结构。

float + overflow

```
<div class="parent">
  <div class="left">
    <p>left</p>
  </div>
  <div class="right">
    <p>right</p>
    <p>right</p>
  </div>
</div>

<style>
  .left {
    float: left;
    width: 100px;
  }
  .right {
    overflow: hidden;
  }
</style>
```

设置 `overflow: hidden` 会触发 BFC 模式（Block Formatting Context）块级格式化文本。BFC 中的内容与外界的元素是隔离的。

优点

- 样式简单

缺点

- 不支持 IE 6

table

```
<div class="parent">
  <div class="left">
    <p>left</p>
  </div>
  <div class="right">
    <p>right</p>
    <p>right</p>
  </div>
</div>

<style>
  .parent {
    display: table;
    width: 100%;
    table-layout: fixed;
  }
  .left {
    display: table-cell;
    width: 100px;
  }
  .right {
    display: table-cell;
    /*宽度为剩余宽度*/
  }
</style>
```

`table` 的显示特性为每列的单元格宽度合一定等与表格宽度。 `table-layout: fixed;` 可加速渲染，也是设定布局优先。

NOTE： `table-cell` 中不可以设置 `margin` 但是可以通过 `padding` 来设置间距。

flex

```
<div class="parent">
  <div class="left">
    <p>left</p>
  </div>
  <div class="right">
    <p>right</p>
    <p>right</p>
  </div>
</div>

<style>
  .parent {
    display: flex;
  }
  .left {
    width: 100px;
    margin-left: 20px;
  }
  .right {
    flex: 1;
    /*等价于*/
    /*flex: 1 1 0;*/
  }
</style>
```

NOTE： `flex-item` 默认为内容宽度。

缺点

- 低版本浏览器兼容问题
- 性能问题，只适合小范围布局。

两列定宽，一列自适应

多列定宽的实现可以更具单列定宽的例子进行修改与实现。




```
<div class="parent">
  <div class="left">
    <p>left</p>
  </div>
  <div class="center">
    <p>center<p>
  </div>
  <div class="right">
    <p>right</p>
    <p>right</p>
  </div>
</div>

<style>
  .left, .center {
    float: left;
    width: 100px;
    margin-right: 20px;
  }
  .right {
    overflow: hidden;
    /*等价于*/
    /*flex: 1 1 0;*/
  }
</style>
```

多列定宽的实现可以更具单列定宽的例子进行修改与实现。

一列不定宽加一列自适应



不定宽的宽度为内容决定，下面为可以实现此效果的方法：

- `float` + `overflow` ，此方法在 IE6 中有兼容性问题
- `table` ，此方法在 IE6 中有兼容性问题
- `flex` ，此方法在 IE9及其以下版本中有兼容性问题

多列不定宽加一列自适应

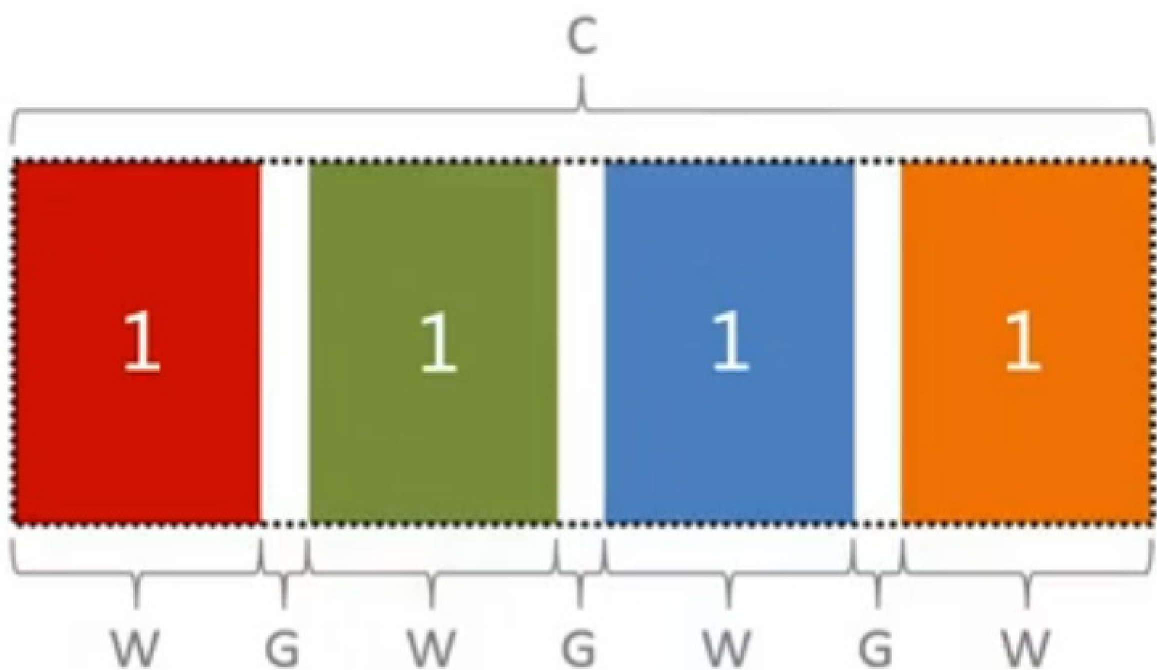


其解决方案同一列不定宽加一列自适应相仿。

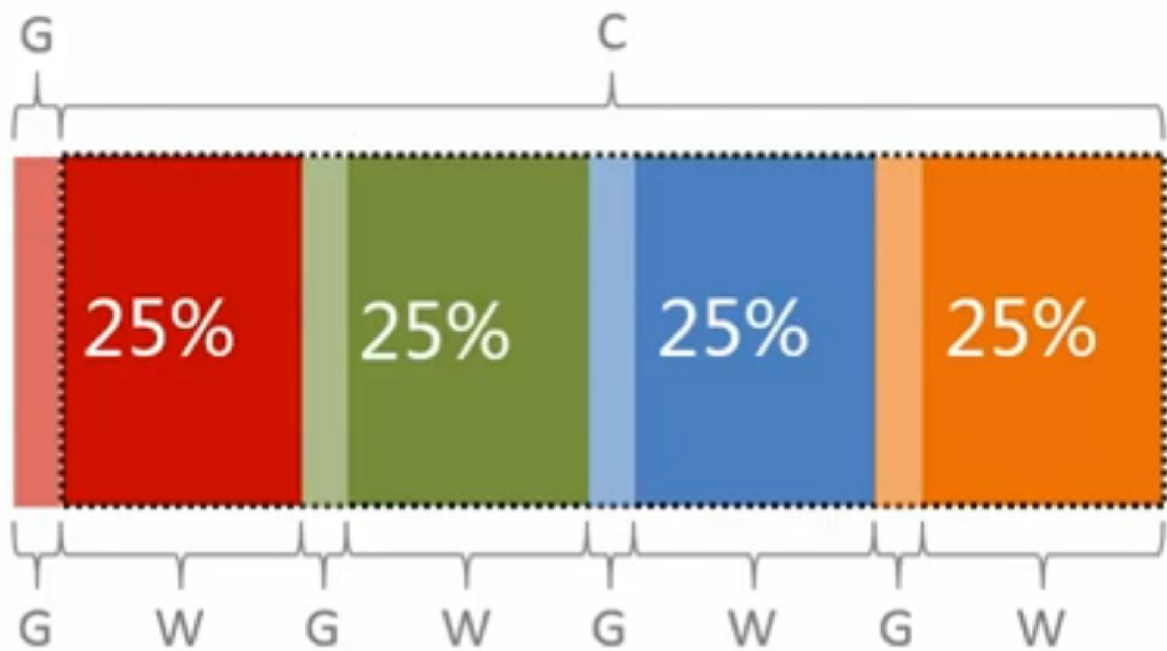
多列等分布局



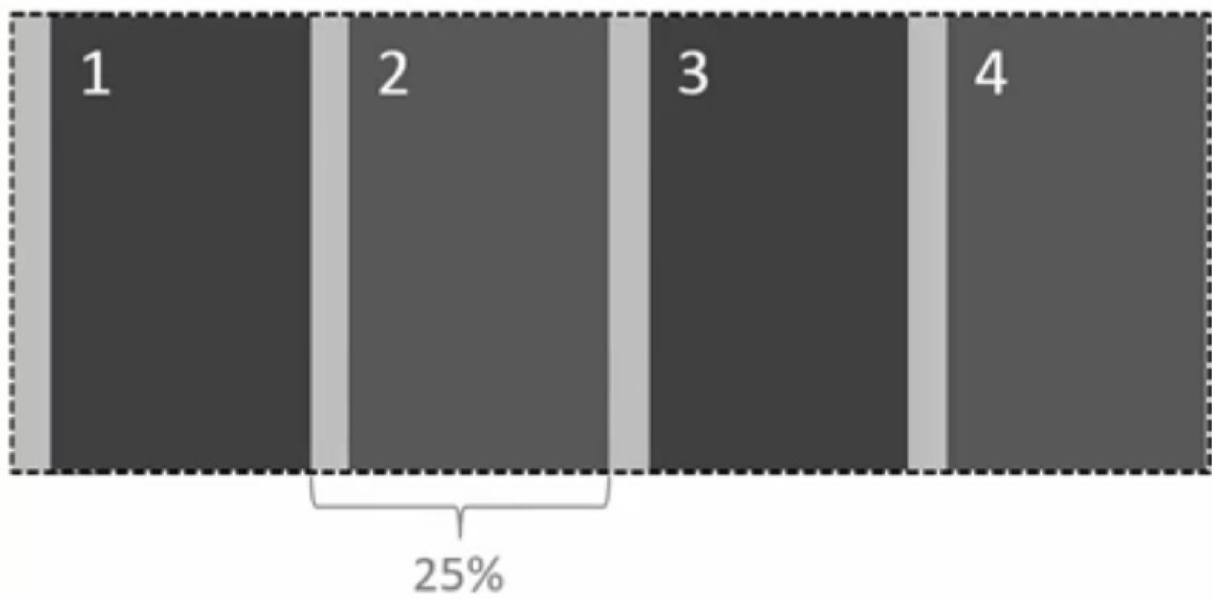
每一列的宽度和间距均相等，下面为多列等分布局的布局特定。



父容器宽度为 C，`C = W * N + G * N - G` => `C + G = (W + G) * N` 。



float



```
<div class="parent">
  <div class="column">
    <p>1</p>
  </div>
  <div class="column">
    <p>2</p>
  </div>
  <div class="column">
    <p>3</p>
  </div>
  <div class="column">
    <p>4</p>
  </div>
</div>
<style media="screen">
  .parent {
    margin-left: -20px;
  }
  .column {
    float: left;
    width: 25%;
    padding-left: 20px;
    box-sizing: border-box;
  }
</style>
```

NOTE：此方法可以完美兼容 IE8 以上版本。 NOTE+：此方法结构和样式具有耦合性。

table

```
<div class='parent-fix'>
  <div class="parent">
    <div class="column">
      <p>1</p>
    </div>
    <div class="column">
      <p>2</p>
    </div>
    <div class="column">
      <p>3</p>
    </div>
    <div class="column">
      <p>4</p>
    </div>
  </div>
</div>

<style media="screen">
  .parent-fix {
    margin-left: -20px;
  }
  .parent {
    display: table;
    width: 100%;
    /*可以布局优先，也可以单元格宽度平分在没有设置的情况下*/
    table-layout: fixed;
  }
  .column {
    display: table-cell;
    padding-left: 20px;
  }
</style>
```

NOTE：缺点是多了文本结果

flex

```
<div class="parent">
  <div class="column">
    <p>1</p>
  </div>
  <div class="column">
    <p>2</p>
  </div>
  <div class="column">
    <p>3</p>
  </div>
  <div class="column">
    <p>4</p>
  </div>
</div>


<style media="screen">
  .parent {
    display: flex;
  }
  .column {
    /*等价于 flex: 1 1 0;*/
    flex: 1;
  }
  .column+.column {
    margin-left: 20px;
  }
</style>
```

NOTE： `flex` 的特性为分配剩余空间。 NOTE+：兼容性有问题。

#



table

`table` 的特性为每列等宽，每行等高可以用于解决此需求。

```
<div class="parent">
  <div class="left">
    <p>left</p>
  </div>
  <div class="right">
    <p>right</p>
    <p>right</p>
  </div>
</div>

<style>
  .parent {
    display: table;
    width: 100%;
    table-layout: fixed;
  }
  .left {
    display: table-cell;
    width: 100px;
  }
  .right {
    display: table-cell;
    /*宽度为剩余宽度*/
  }
</style>
```

flex

```
<div class="parent">
  <div class="left">
    <p>left</p>
  </div>
  <div class="right">
    <p>right</p>
    <p>right</p>
  </div>
</div>

<style>
  .parent {
    display: flex;
  }
  .left {
    width: 100px;
    margin-left: 20px;
  }
  .right {
    flex: 1;
    /*等价于*/
    /*flex: 1 1 0;*/
  }
</style>
```

NOTE：flex 默认的 `align-items` 的值为 `stretch` 。

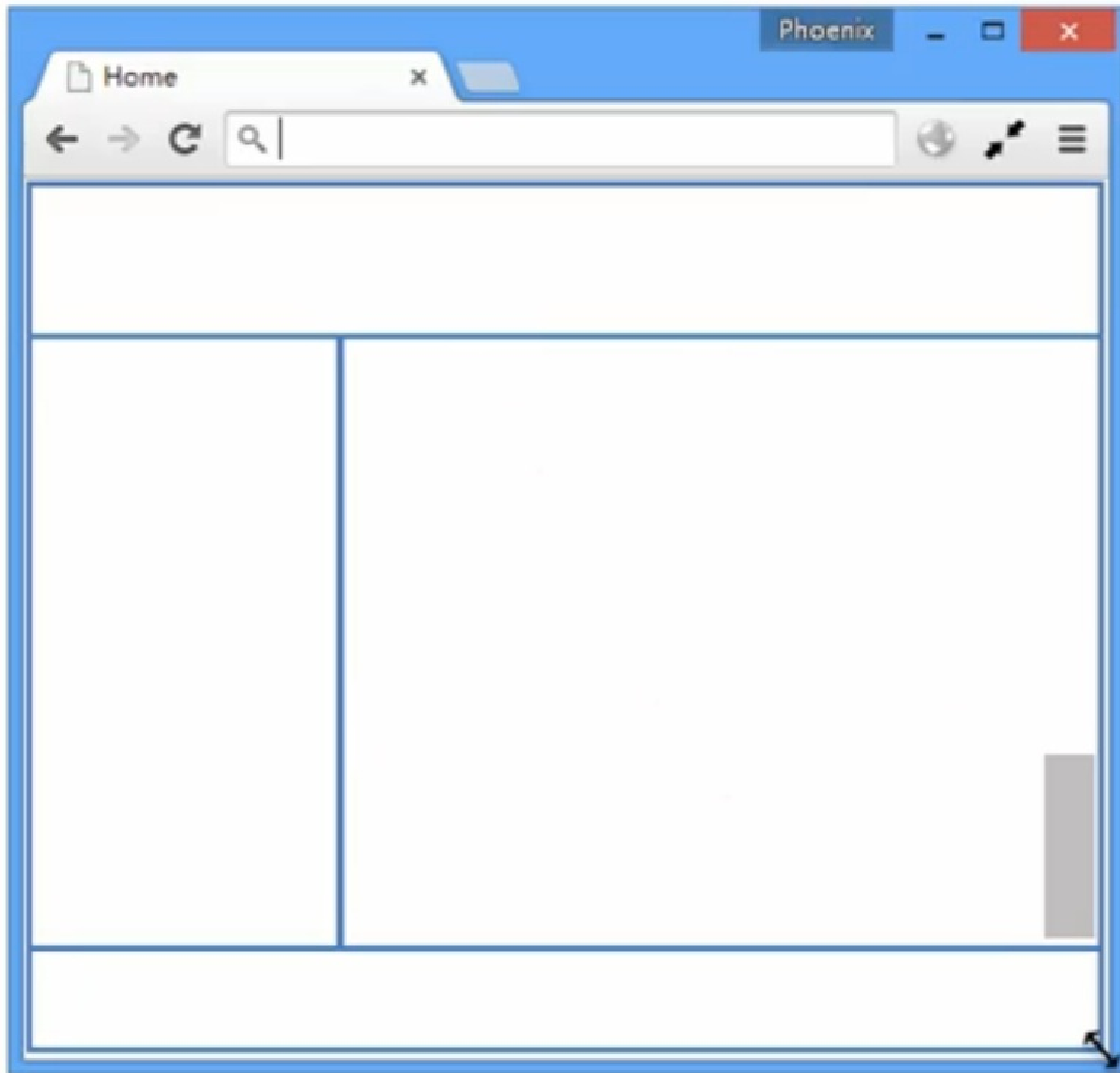
float

```
<div class="parent">
  <div class="left">
    <p>left</p>
  </div>
  <div class="right">
    <p>right</p>
    <p>right</p>
  </div>
</div>

<style>
  .parent {
    overflow: hidden;
  }
  .left,
  .right {
    padding-bottom: 9999px;
    margin-bottom: -9999px;
  }
  .left {
    float: left;
    width: 100px;
    margin-right: 20px;
  }
  .right {
    overflow: hidden;
  }
</style>
```

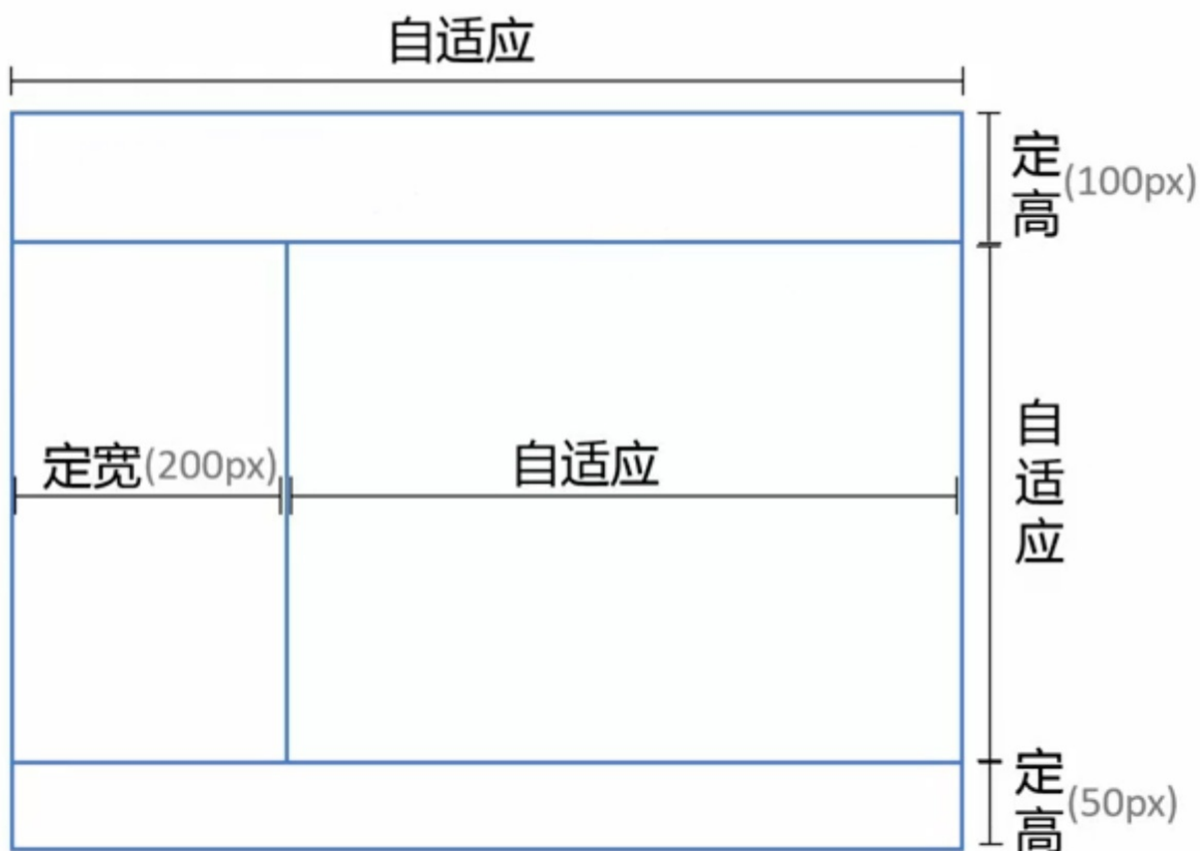
NOTE：此方法为伪等高（只有背景显示高度相等），左右真实的高度其实不相等。 NOTE+：此方法兼容性较好。

全屏布局



例如管理系统，监控与统计平台均广泛的使用全屏布局。

定宽需求



实现方案

- Position 常规方案
- Flex CSS3 新实现

Position

```
<div class="parent">
  <div class="top"></div>
  <div class="left"></div>
  <div class="right">
    /*辅助结构用于滚动*/
    <div class="inner"></div>
  </div>
  <div class="bottom"></div>
</div>
<style>
  html,
  body,
  .parent {
```

```
    height: 100%;
    /*用于隐藏滚动条*/
    overflo: hidden;
  }
  .top {
    /*相对于 body 定位*/
    position: absolute;
    top: 0;
    left: 0;
    right: 0;
    height: 100px;
  }
  .left {
    position: absolute;
    left: 0;
    top: 100px;
    bottom: 50px;
    width: 200px;
  }
  .right {
    position: absolute;
    left: 200px;
    right: 0;
    top: 100px;
    bottom: 50px;
    overflow: auto;
  }
  .right .inner {
    /*此样式为演示所有*/
    min-height: 1000px;
  }
  .bottom {
    position: absolute;
    left: 0;
    right: 0;
    bottom: 0;
    height: 50px;
  }
</style>
```

Position 兼容

此方法不支持 IE6 可以使用下面的方法解决兼容问题。

```
<div class="g-hd"></div>
<div class="g-sd"></div>
<div class="g-mn"></div>
<div class="g-ft"></div>
<style>
  html,
  body {
    width: 100%;
    height: 100%;
    overflow: hidden;
    margin: 0;
  }

  html {
    _height: auto;
    _padding: 100px 0 50px;
  }

  .g-hd,
  .g-sd,
  .g-mn,
  .g-ft {
    position: absolute;
    left: 0;
  }

  .g-hd,
  .g-ft {
    width: 100%;
  }

  .g-sd,
  .g-mn {
    top: 100px;
    bottom: 50px;
    _height: 100%;
```

```
    overflow: auto;
  }

  .g-hd {
    top: 0;
    height: 100px;
  }

  .g-sd {
    width: 300px;
  }

  .g-mn {
    _position: relative;
    left: 300px;
    right: 0;
    _top: 0;
    _left: 0;
    _margin-left: 300px;
  }

  .g-ft {
    bottom: 0;
    height: 50px;
  }
</style>
```

Flex

```
<div class="parent">
  <div class="top"></div>
  <div class="middle">
    <div class="left"></div>
    <div class="right">
      <div class="inner"></div>
    </div>
  </div>
  <div class="bottom"></div>
</div>
```

```
<style media="screen">
  html,
  body,
  parent {
    height: 100%;
    overflow: hidden;
  }

  .parent {
    display: flex;
    flex-direction: column;
  }

  .top {
    height: 100px;
  }

  .bottom {
    height: 50px;
  }

  .middle {
    // 居中自适应
    flex: 1;
    display: flex;
    /*flex-direction: row 为默认值*/
  }

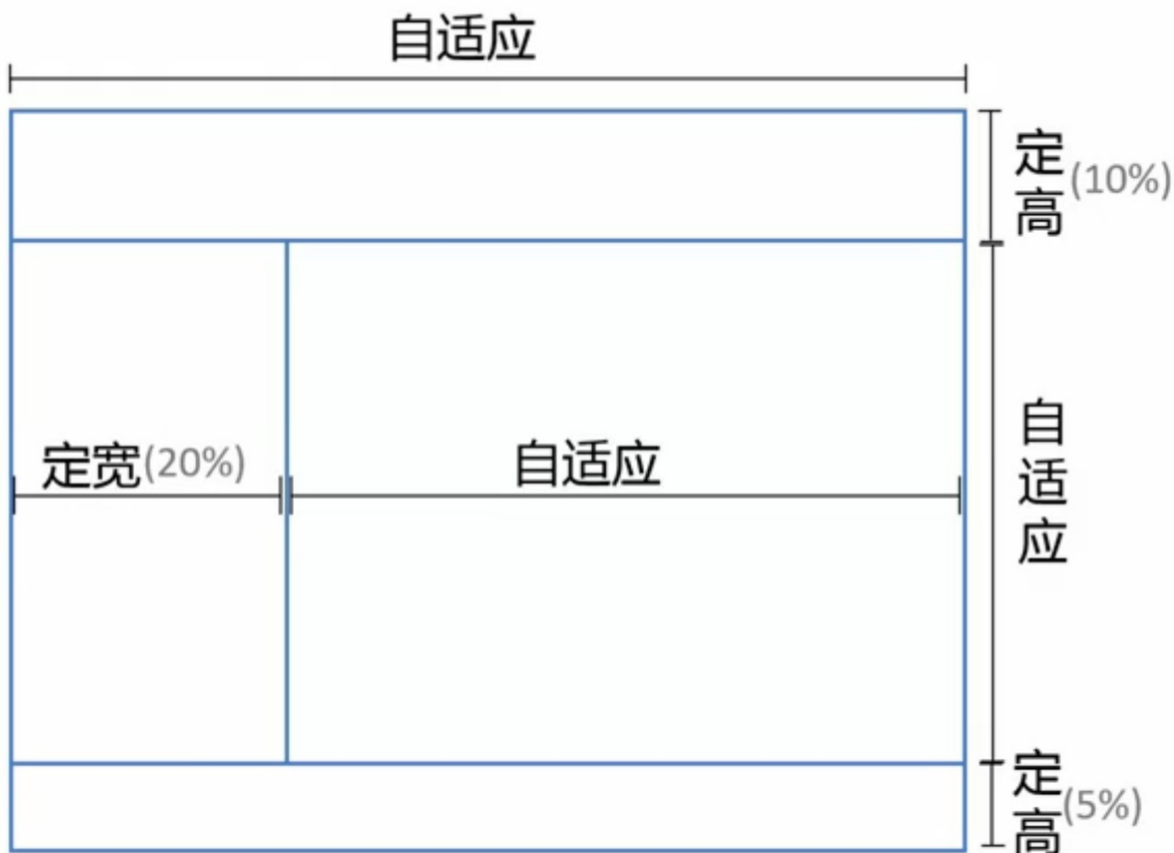
  .left {
    width: 200px;
  }

  .right {
    flex: 1;
    overflow: auto;
  }
  .right .inner {
    min-height: 1000px;
  }
</style>
```

Flex 兼容性

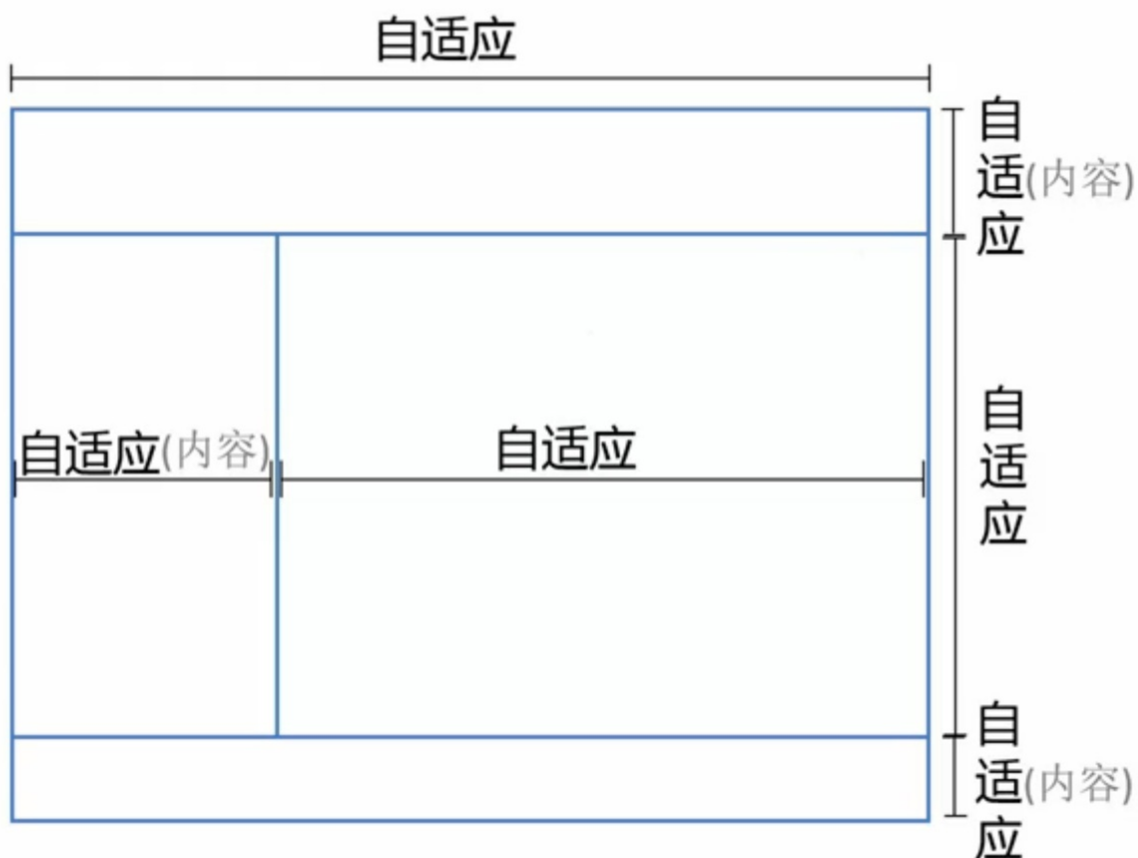
CSS3 中的新概念所有 IE9 及其也行版本都不兼容。

百分比宽度需求



只需把定宽高（`px` 为单位的值）的实现改成百分比（%）既可。

内容自适应



只有右侧栏占据剩余位置，其余空间均需根据内容改变。 所以 Postion 的定位方法不适合实现此方案。下面列出了两种布局方案：

- Flex
- Grid，W3C 草案并不稳定，浏览器支持也并不理想

Flex

只有不为宽高做出限制，既可对其中的内容做出自适应的布局。

```
<div class="parent">
  <div class="top"></div>
  <div class="middle">
    <div class="left"></div>
    <div class="right">
      <div class="inner"></div>
    </div>
  </div>
  <div class="bottom"></div>
</div>

<style media="screen">
  html,
  body,
  parent {
    height: 100%;
    overflow: hidden;
  }

  .parent {
    display: flex;
    flex-direction: column;
  }

  .middle {
    // 居中自适应
    flex: 1;
    display: flex;
    /*flex-direction: row 为默认值*/
  }

  .right {
    flex: 1;
    overflow: auto;
  }
  .right .inner {
    min-height: 1000px;
  }
</style>
```

方案比较

方案	兼容性	性能	自适应
Position	好	好	部分自适应
Flex	较差	差	可自适应
Grid	差	较好	可自适应

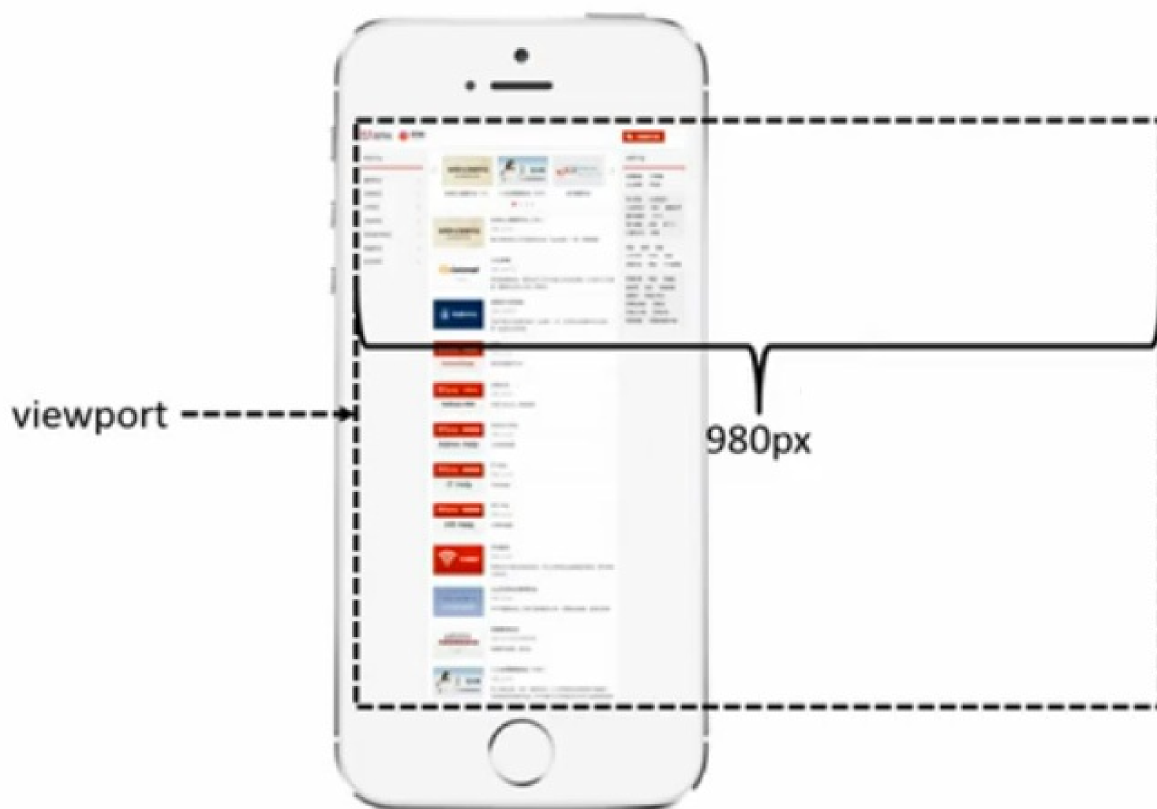
响应式布局

多屏的环境让我们不得不考虑网络内容在各个尺寸中的表现， 均可正常访问和极佳的用户体验。

响应式布局可以更具屏幕尺子的大小对内容和布局做出适当的调成， 从而提供更好的用户感受。也是因为响应式布局的出现， 开发者也无需对不同尺寸设备而特殊定制不同的页面， 这大大降低了开发成本和缩短了开发时间。

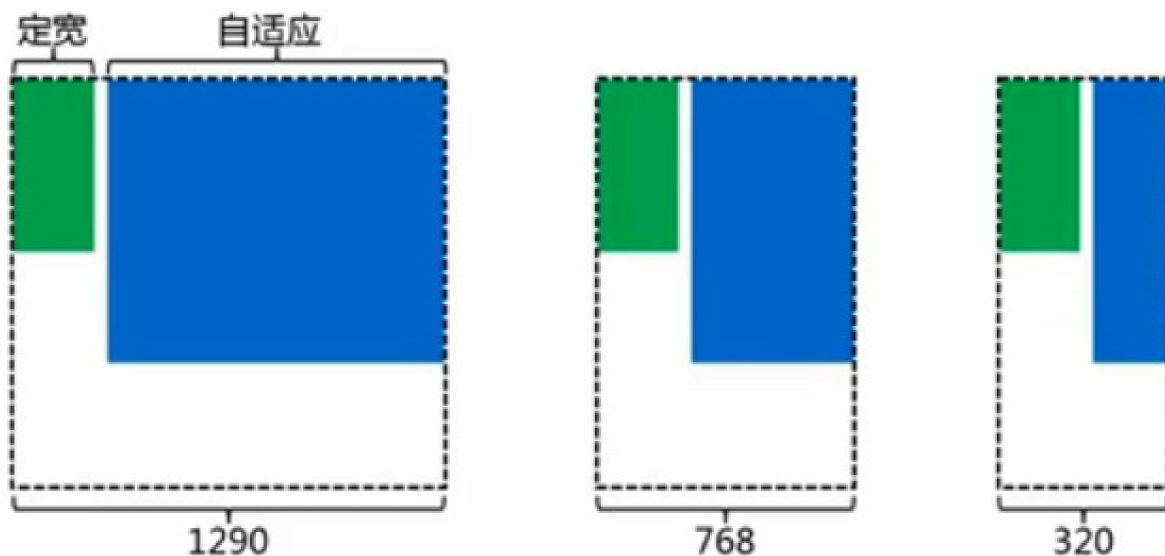
这样的方法也同样存在着缺点。 缺点是同样的资源被加载，但因为展示平台所限并不能全部显示。

View Port




```
<meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=no">
```

针对不同尺寸的屏幕进行开发，因少使用定宽而使用自适应单位。



需求会更具具体设备而产生变化。（例如布局方式发生了变化）

```
@media screen and (max-width: 320px) {
  /* 视窗宽度小于等于 320px */
}
@media screen and (min-width: 320px) {
  /* 视窗宽度大于等于 320px */
}
@media screen and (min-width: 320px) and (max-width: 1000px){
  /* 视窗宽度大于等于 320px 且小于等于 1000px */
}
```

页面优化

页面优化可以提升页面的访问速度从而提高用户体验，优化的页面可以更好的提升 SEO 的效果同时也可以提高代码的可读性和维护性。

从下面的几个方面可以进行页面的优化：

- 减少请求数
    - 图片合并
    - CSS 文件合并
    - 减少内联样式
    - 避免在 CSS 中使用 import
- 减少文件大小
    - 选择适合的图片格式
    - 图片压缩
    - CSS 值缩写（Shorthand Property)
    - 文件压缩
- 页面性能
    - 调整文件加载顺序
    - 减少标签数量
    - 调整选择器长度
    - 尽量使用 CSS 制作显示表现
- 增强代码可读性与可维护性
    - 规范化
    - 语义化
    - 模块化

减少请求

请求数与网页加载时长有直接的关系。所以对于图标可以使用 Sprite 来减少小图标的请求数，对于文本则可以通过合并缩小。（避免使用 import 引入 CSS 文件，并且请求是同步请求）

减少文件大小

页面上最大的流量产生与多媒体（视频或图片）所以为了减少文件大小，开发者需要使用合适的媒体格式并对文件进行压缩。

页面性能

页面文件的加载顺序自上而下，样式则需要放置于最顶部，脚本则放置于底部（因为 JavaScript 的加载会阻塞页面的绘制）。

减少标签的数量也可以起到提升性能的作用，缩短 CSS 选择器的层级来提高性能。减少使用消耗性能的样式属性例如下面的这些：

- expression `.class{width: expression(this.width > 100?'100px':'auto')}`
- filter `.class{filter:alpha(opacity=50)}`
- border-radius
- box-shadow
- gradients

页面中所使用的图片尺子应该与现实尺寸相符否则在图标下载后需要重绘导致性能下降。

能使用样式（使用 CSS 的类名）实现的交互就不使用脚本（需要重绘导致多次页面渲染）来实现。

可读性与可维护性

开发之前需要明确规范，尤其是对人协作时。使用 HTML5 语义化的标签来制作页面，同样也适用于样式选择器的 ID 与类名。在使用开发中的奇技淫巧的适合需要深思是否需要使用。模块化的制作页面和样式，提高可复用性并减少文件大小。

注释注释注释，在代码中添加注释，利人利己。

页面模块化

规范

在具体谈论规范的之前，可以下去查看下各大网络公司的前端开发规范（Developemnt Style Guide）例如谷歌，Facebook 或者 Dropbox。从而更好的理解开发规范在实际应用中和多人协作中的重要性。

不同开发者在开发过程中使用不同的代码风格会直接的提升在之后的开发和维护的成本和难度，对前端开发来说更是尤为突出。这时使用代码规范来约束开发者的编码风格就可以大体解决这些问题。规范的制订应从下面的几个方面来开始考虑：

- 文件规范
- 注释规范
- 命名规范
- 书写规范
- 其他规范

文件规范

文件规范又可以从三个方面入手，分类，引入，以及文本本身的内容。

- 分类（分类可分为通用类和业务类。通用类有第三方的库，团队开发的通用模块或者通用样式。业务类则有不同业务所对应的特定模块。）

- 引入

    - CSS （引入文件则需尽少的使用行内样式）
    - JavaScript （文件名的约束，以及编码设置通常使用 `utf-9` ）

注释规范

注释可使用块状，单行注释和行内注释，需要统一缩进等细节要求。

命名规范

例如 CSS 选择器的命名规范

- 分类命名 （例如 `g-header` 来给布局类的样式设置命名空间来防止样式污染， `m-header` 来制定模块类的样式）

- 命名格式（大小写的规定，建议使用小写并使用 `-` 分隔，也许控制选择器的长度避免过长的样式选择器名称但不可失去选择器语义）
- 语义化命名（以内容的语义来给选择器命名）

书写规范

这里使用 CSS 的书写规范来做示例，可以参考下面的约束：

- 单行与多行（单行与多行的 CSS 书写格式，使用多行！）
- 空格与分号（使用空格进行缩进并保留最后一个属性的分号）
- 书写顺序（更具显示的重要性来安排可参照下表）
- Hack 方式（三思而后行）
- 值格式（例如颜色值的格式以及引用中是否使用引号）

→	显示属性	自身属性	文本属性和其他修饰
	display	width	font
	visibility	height	text-align
	position	margin	text-decoration
	float	padding	vertical-align
	clear	border	white-space
	list-style	overflow	color
	top	min-width	background

其他规范

这里包括有 HTML 以及图片的规范：

- HTML
  - 文档声明
  - 闭合
  - 属性
  - 层级
  - 注释
  - 大小写
- 图片
  - 文件名称（语言以及长度的规范）
  - 保留源文件

- 图片合并

模块化

模块化是一系列相关的结果组成的整体，这部分具备独立存在的意义不单纯只是表现。

在开发模块化是需要注意的一些步骤（以 CSS 模块化为例）：

- 为模块分类命名（ `m-module-name` ）
- 以一个主选择器作为开头（模块跟节点）
- 使用以主选择器开头的后代选择器（模块子节点）

```
<!- NAV MODULE -->
<div class="m-nav">
    <ul>
        <li class="z-crt"><a href="#">index</a></li>
        <li><a href="#">link1</a></li>…
    </ul>
</div>
<!-- /NAV MODULE -->
```

index link1 link2

```
/* 导航模块 */
.m-nav{}/* 模块容器 */
.m-nav li,.m-nav a{}
.m-nav li{}
.m-nav a{}
.m-nav .z-crt a{}/* 交互状态变化 */
```

```
<!- NAV-1 MODULE -->
<div class="m-nav m-nav-1">
    <ul>
        <li class="z-crt"><a href="#">index</a></li>
        <li><a href="#">link1</a></li>…
    </ul>
    <a class="btn">Login</a>
</div>
<!-- /NAV-1 MODULE -->
```

index link1 link2 Login

```
/* 导航模块扩展 */
.m-nav-1{}
.m-nav-1 a{border-radius:5px;}
.m-nav-1 .btn{}
```

```
<div class="m-slides">
    <ol>
        <li class="slide z-crt"><img src="#"></li>
        <li class="slide"><img src="#"></li>
        <li class="slide"><img src="#"></li>
        <li class="slide"><img src="#"></li>
    </ol>
    <span class="ctrl"><i class="z-crt"></i><i></i><i></i><i></i></span>
</div>
```

```
/* 未模块化样式 */
.slides{}
.slide{}
.slide img{}
li.z-crt{}
.ctrl{}
.ctrl i{}
i.z-crt{}
```

```
/*  模块化样式 */
.m-slides{}
.m-slides .slide{}
.m-slides .slide img{}
.m-slides li.z-crt{}
.m-slides .ctrl{}
.m-slides .ctrl i{}
.m-slides .ctrl i.z-crt{}
```

模块化可以利于多人开发，便于扩展，当然也可以提高代码的可读性与可维护性。

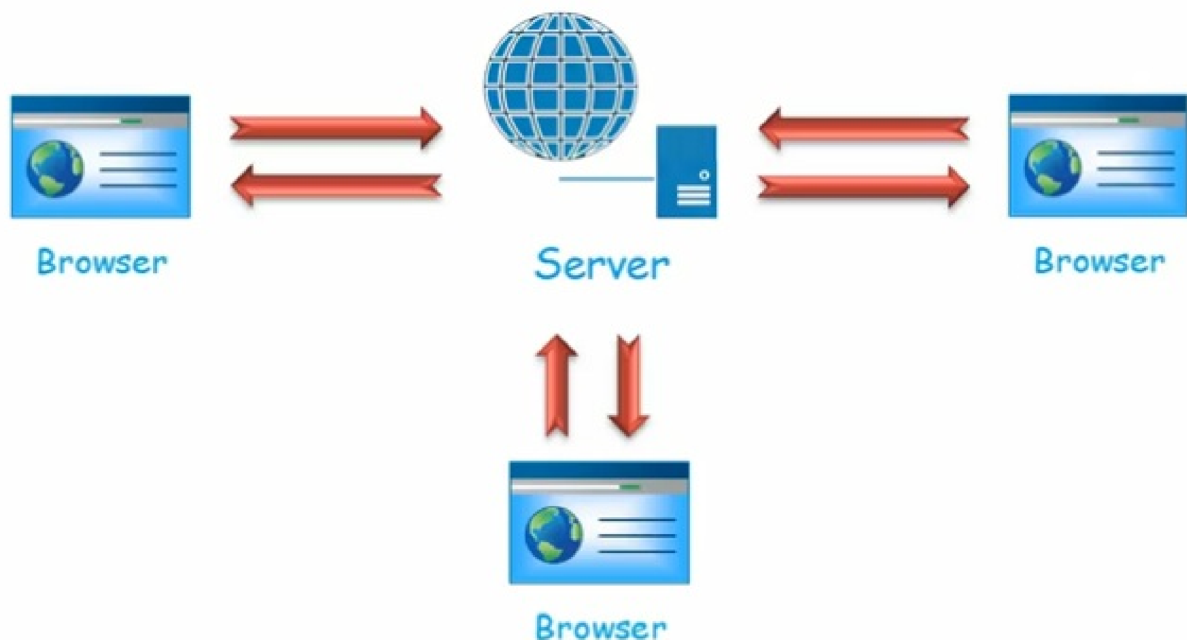
产品前端架构

专注于引导前端工程师去主导团队高效协作，去引领团队规范化、工程化构建复杂系统的实践过程。

本章节的核心内容包括多角色低耦合的协作流程、协作规范、 多角度分析选择适合具体项目的技术解决方案、工程化的版本管理、系统构建案例分析。

协作流程

Web 系统



Web 系统部署在服务器上为提出不同需求的外部客户端服务。

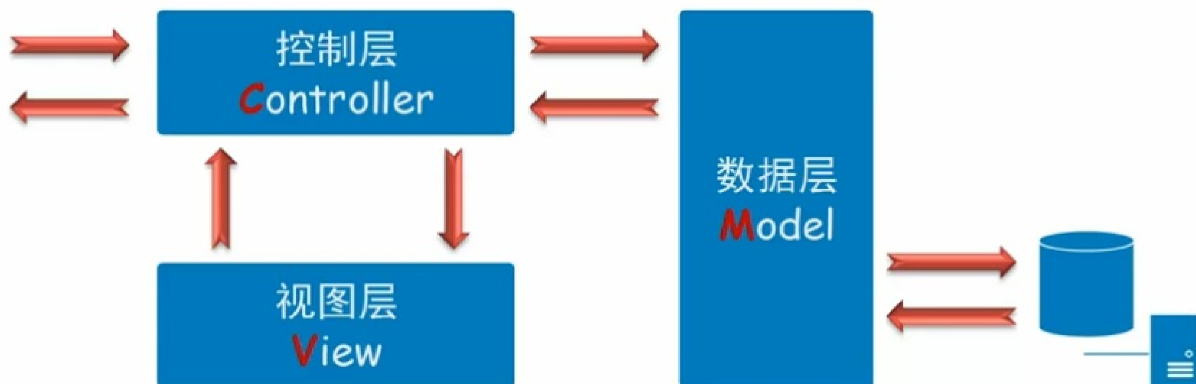
MVC (Model, View, Controller) 它们分别代表数据层，视图层， 和控制层。



- 数据层，封装数据管理操作（例如数据的 CRUD ）
- 视图层，展示数据模型提供人机交互
- 控制层，处理用户请求，委托数据层，选择视图层进行展示

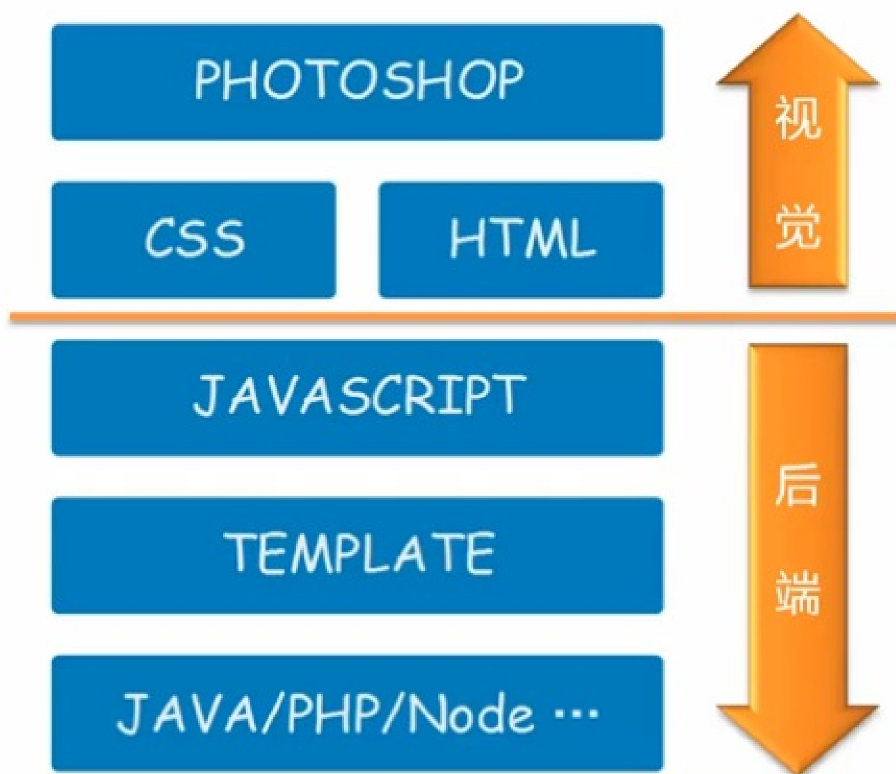
NOTE：CRUD 代表 Create、Read、Update、Delete。

下面以用户请求页面为例：



1. 客户端发送请求，服务器控制层接受到请求
2. 请求数据层获取数据，返回控制层
3. 控制层根据数据选择合适的视图层进行展示
4. 视图层生成页面代码，返回控制层
5. 控制层返回客户端进行展示

技术栈全览

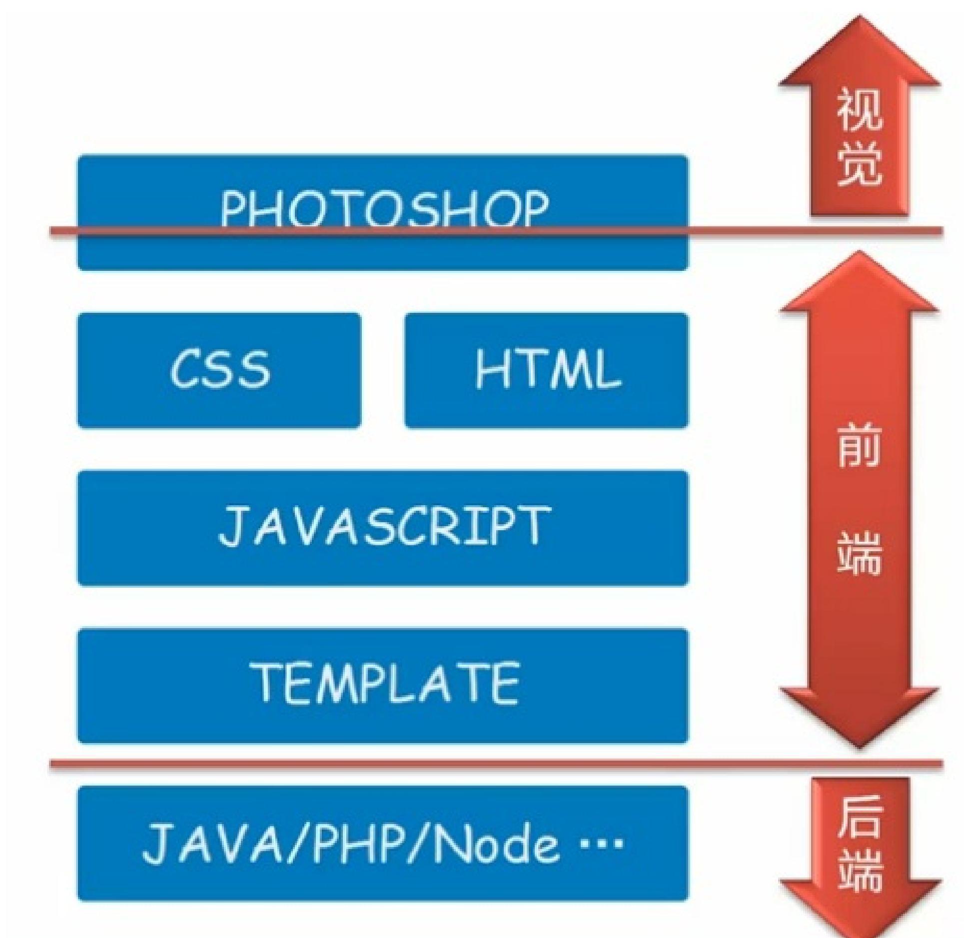


- Photoshop，获取图片资源
- CSS 与 HTML，制作页面
- JavaScript 前端交互逻辑
- Template 结构与内容分离整合
- Java/PHP/Node 后端逻辑

弊端

- 后期维护性差（相同内容，不同形成存在）
- 专业化程度低
- 需求响应速度慢

前端工程师新责任



随着视图层也会存在业务逻辑的需求，前端工程师也会参与到业务逻辑的实现中去。 这样使前端工程师可以在后期转换成全栈工程师（Fullstack Developer)。

角色定义

完成一个 Web 系统需要至少以下三种角色：

- 视觉工程师，视觉稿到交互原型的转化
- 前端工程师，实现系统前端交互逻辑
- 后端工程师，系统后端业务逻辑

前端工程师



前端部分又可再细分为页面工程师和前端工程师**。前者更注重和视觉工程师的协作， 后者则更多的与后端工程师进行协作。

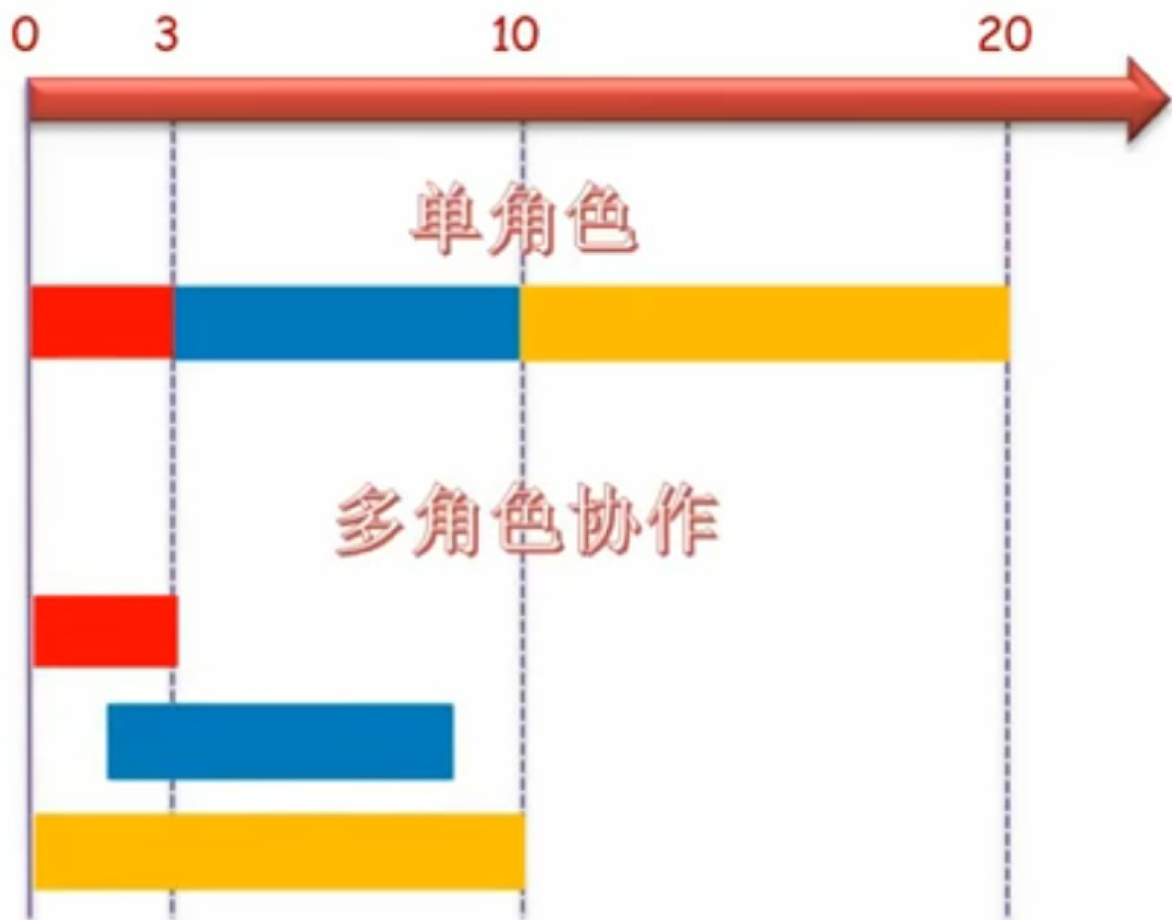
页面工程师

- 精通切图技术 （Photoshop，Sketch）
- 精通页面制作 （CSS，HTML）
- 熟悉前端开发技术 （JavaScript，Template）
- 了解后端开发技术 （Java，Node）

前端工程师

- 精通页面制作 （CSS，HTML）
- 精通前端开发技术 （JavaScript，Template）
- 熟悉切图技术 （Photoshop，Sketch）
- 熟悉后端开发技术 （Java，Node）

项目工时分配比

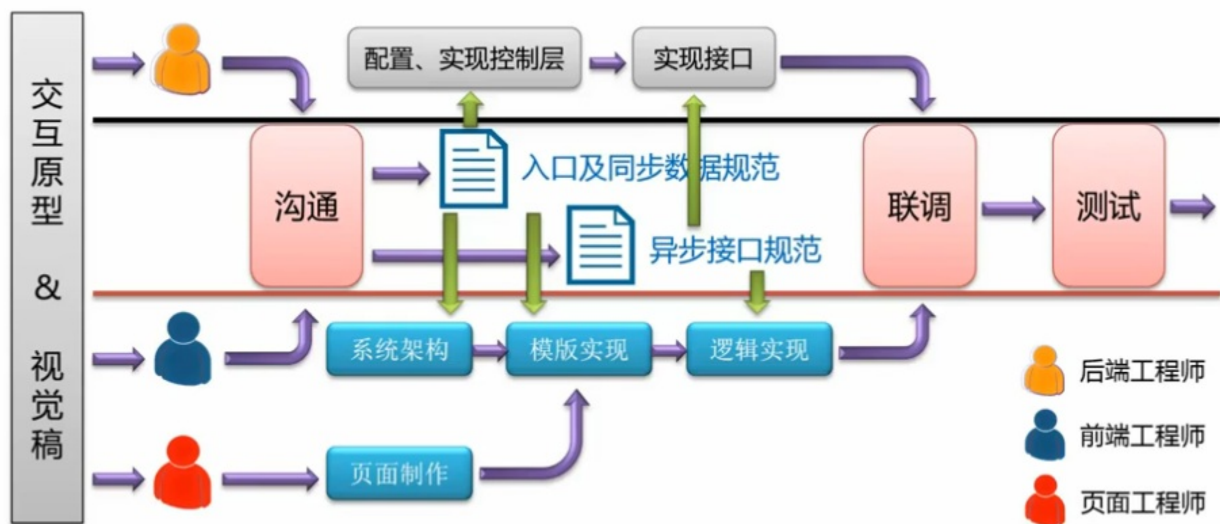


- 页面制作：3天
- 前端逻辑：7天
- 后端逻辑：10天

NOTE：多角色开发会比单人工程增加工作沟通成本。

协作流程

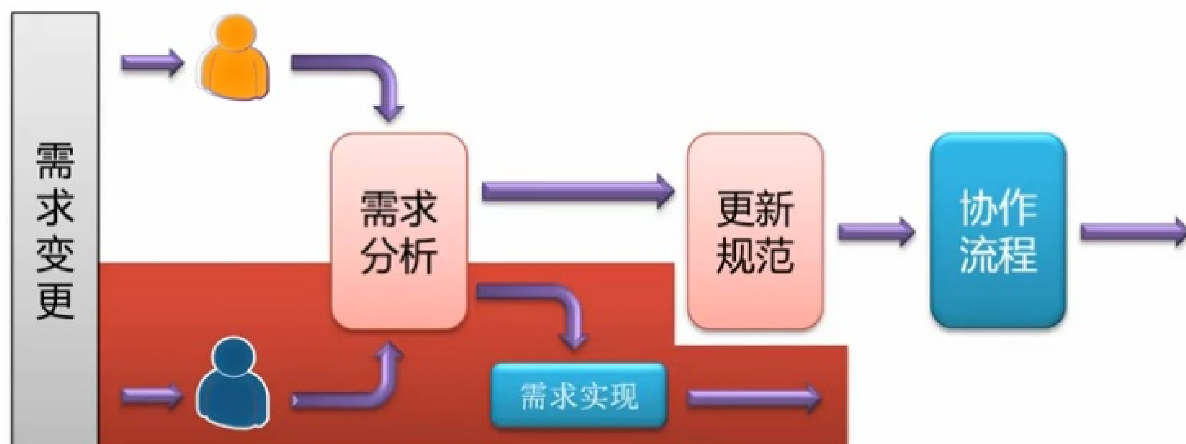
开发过程



按照流程规范可以明确角色和其对应的职责，这样可以大大减少角色间的沟通成本。

- 页面入口规范，定义系统对外可访问入口和配置信息
- 同步数据规范，定义系统对模板文件的预填信息
- 异步接口规范，定义前后端接口信息

维护过程



NOTE：红色路径为不需要改变规范的前提下，响应需求变更。

职责说明

下面将总结各个角色职责具体任务：

页面工程师

- 切图、图片优化
- 页面制作、优化页面效果与结构（适合业务逻辑开发）
- 完成简单的前端业务逻辑开发

前端工程师

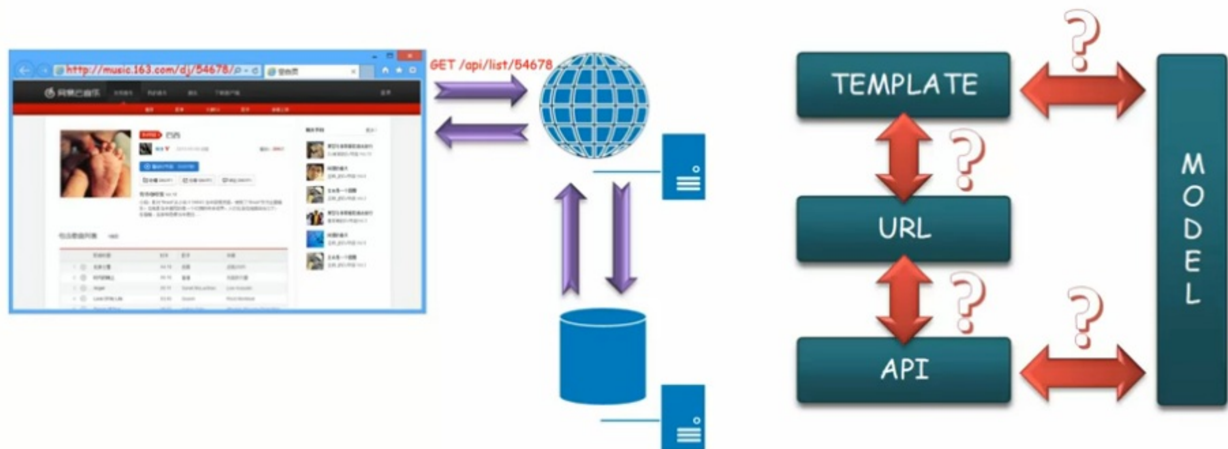
- 主导制定前后端分离规范
- 主导前端联调对接测试
- 系统前端设计架构、满足一定的非功能性需求
- 完成系统前端的业务逻辑实现、优化实现逻辑

后端工程师

- 协助定制前后端分离规范
- 协作前后端联调对接测试
- 完成后端系统框架及业务逻辑实现

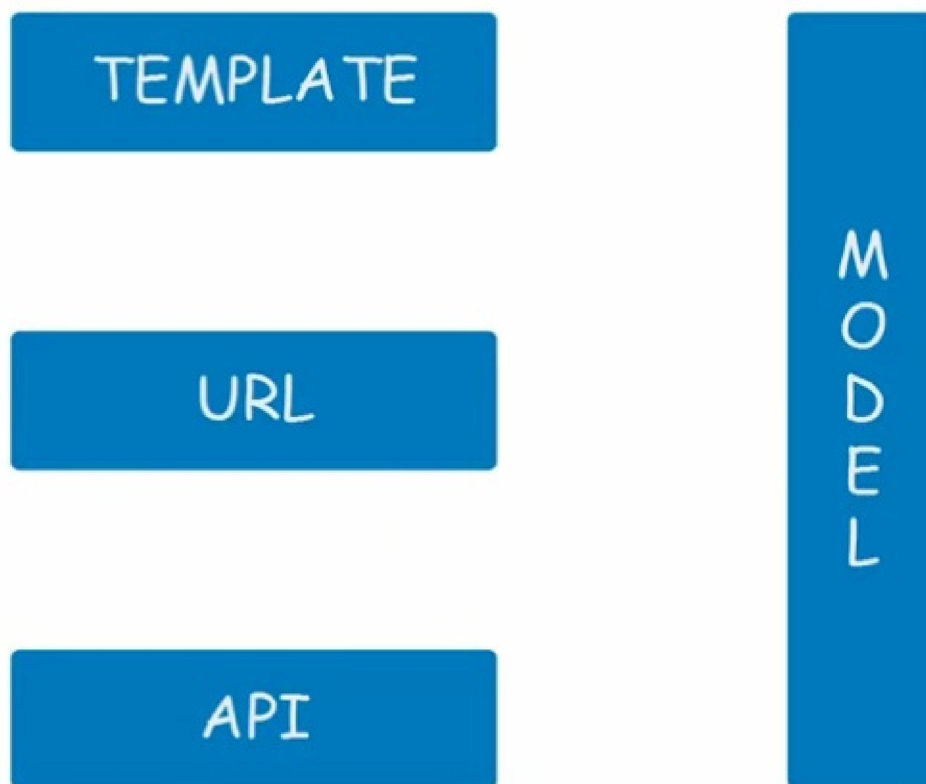
角色与人之间不一定需要一一对应，前端工程师和页面工程师可能是一人。全栈工程师则有能力包揽一切。

接口设计



用户使用 Web 客户端访问 Web 系统，系统在收到请求后执行操作（收集数据模型，选择数据经行组装），将结果返回给客户。

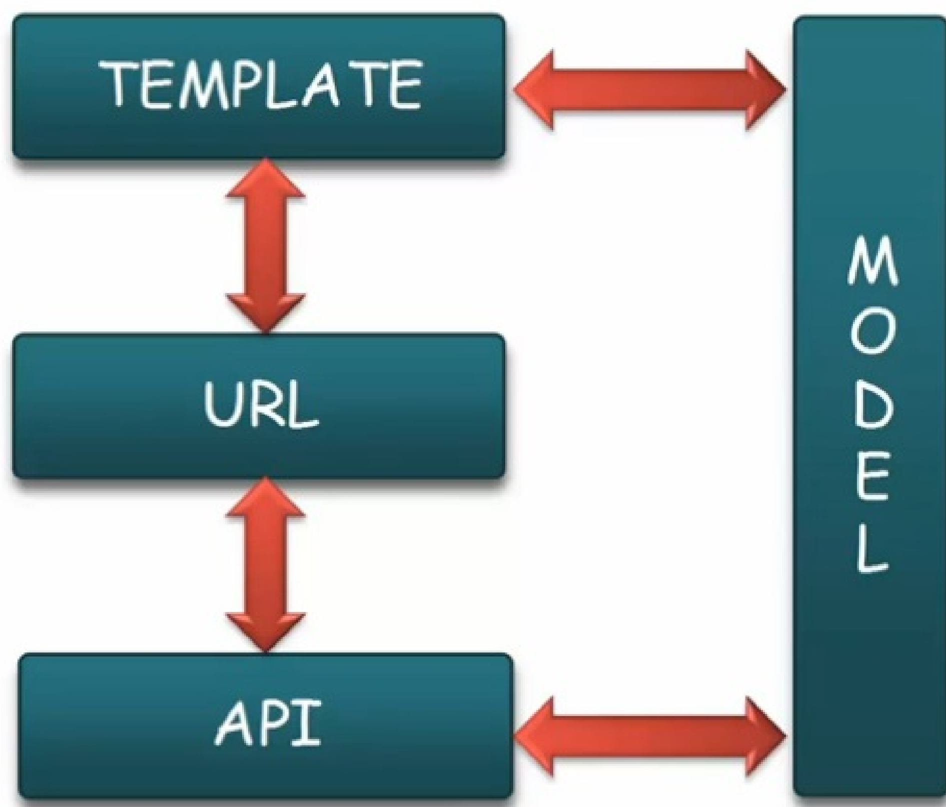
其中包括的元素和关系如下图所示：



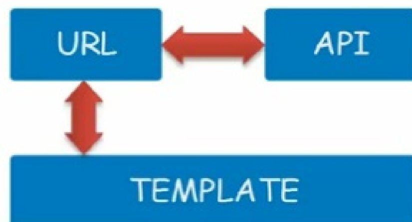
- Template，分离数据模型的页面结构，根据不同的数据模型展现不同的信息
- URL，页面访问地址、页面标示
- API，用于载入异步请求的接口

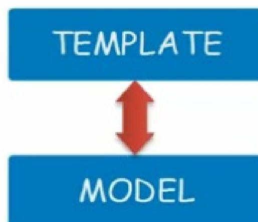
- Model，数据模型，页面模板组装模型和异步请求返回的数据模型

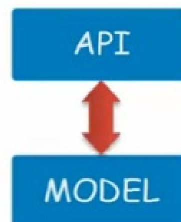
约定



1. URL 与页面模板间的映射，和异步载入内容对应的接口
2. 视图模板和数据模型的对应（数据模型的格式和类型）
3. 异步接口输入输出信息的约定





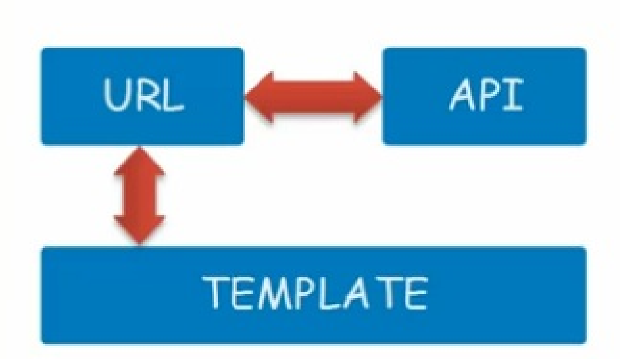


1. 页面入口规范，定义系统对外可访问入口和配置信息（URL、模板、接口）
2. 同步数据规范，定义系统对模板文件的预填信息（模板、数据模型）
3. 异步接口规范，定义前后端接口信息（接口、数据模型）

接口规范

每个规范也会对应若干规定若干规则约定来指导前后端工程师的具体实施。

页面入口规范



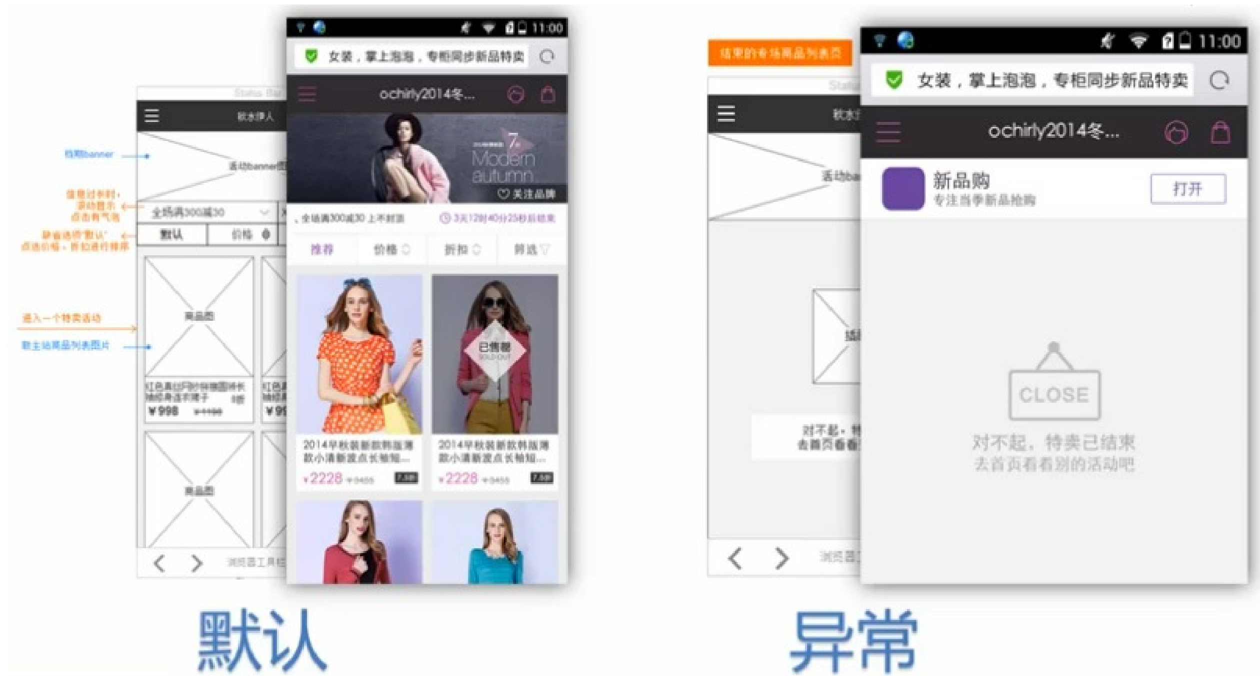
- 基本信息
- 输入参数
- 模板列表
- 接口列表（异步载入数据接口）

页面入口规范（范例）

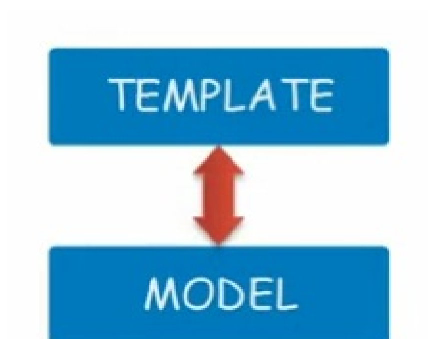
条目	详情
访问地址	`/dj/{id}`
页面描述	节目详情及推荐信息
输入参数	名称：ID；类型：Number；描述：节目标示
异常跳转	异常：500；跳转地址： `/500/`
模板列表	默认： `/template/dj/dj.ftl`
	过期： `/template/dj/over.ftl`
	未找到： `/template/dj/404.ftl`
接口列表	歌曲列表： `/api/dj/tracks/{id}/`
	相关节目： `/api/dj/rec/{id}/`
	收藏节目： `/api/dj/fav/{id}/`
	评论节目： `/api/comments/{id}`

- 页面基本信息，描述页面的主要功能
- 输入参数为访问地址时支持的参数说明
- 异常跳转，为系统全局异常处理

- 模板列表，列举当前页面的相关模板，包括异常（如下图）
- 页面需要的所有异步接口列表



同步数据规范



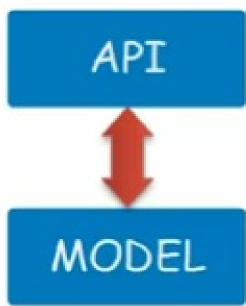
- 基本信息
- 预填数据
- 注入接口

同步数据规范（范例）

条目	详情
模板文件	`/templates/dj/dj.ftl`
模板描述	节目详情及推荐信息模板文件
预填信息	`{"名称":"user", "类型":"User","描述":"登陆用户信息"}`
	`{"名称":"dj", "类型":"Program","描述":"节目信息"}`
	`{"名称":"other", "类型":"String","描述":"其他信息"}`
注入接口	jd.parser
	`{"输入":[["String", "节目信息"], ["Boolean", "是否格式化"]]}`
	`{"输出":["Program", "节目对象"]}`
	dj.api
	dj.api2

- 模板的基本信息
- 预填数据包括全局和页面数据及其类型
- 注入的接口说明（输入输出信息），没有可以不填写

异步接口数据规范



- 基本信息
- 输入信息
- 输出信息

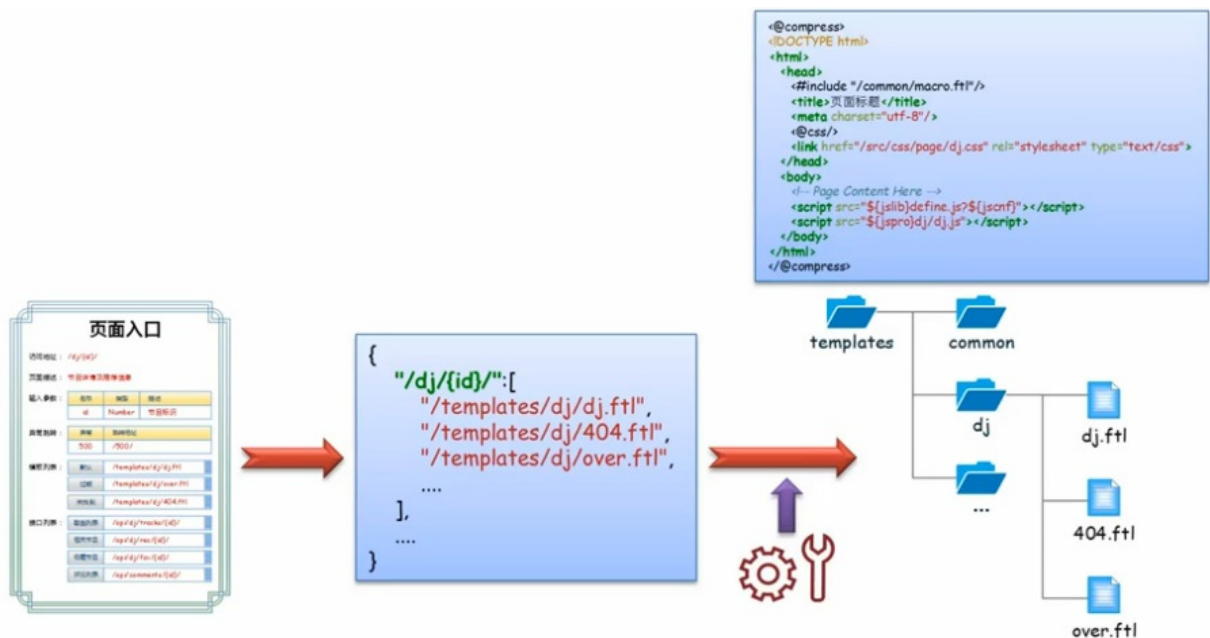
异步接口规范（范例）

条目	详情
请求方式	GET POST
接口地址	api/dj/tracks/{id}/
接口描述	获取指定节目的歌曲列表
输入数据	{"名称":"id", "类型":"Number","描述":"节目标示"}
	{"名称":"offset", "类型":"Number","描述":"节目起始位置"}
	{"名称":"limit", "类型":"Number","描述":"列表数量"}
输出结果	{"名称":" code", "类型":"Number","描述":"请求结果标示"}
	{"名称":"message", "类型":"String","描述":"请求异常信息"}
	{"名称":"result", "类型":"Array","描述":"歌曲列表"}

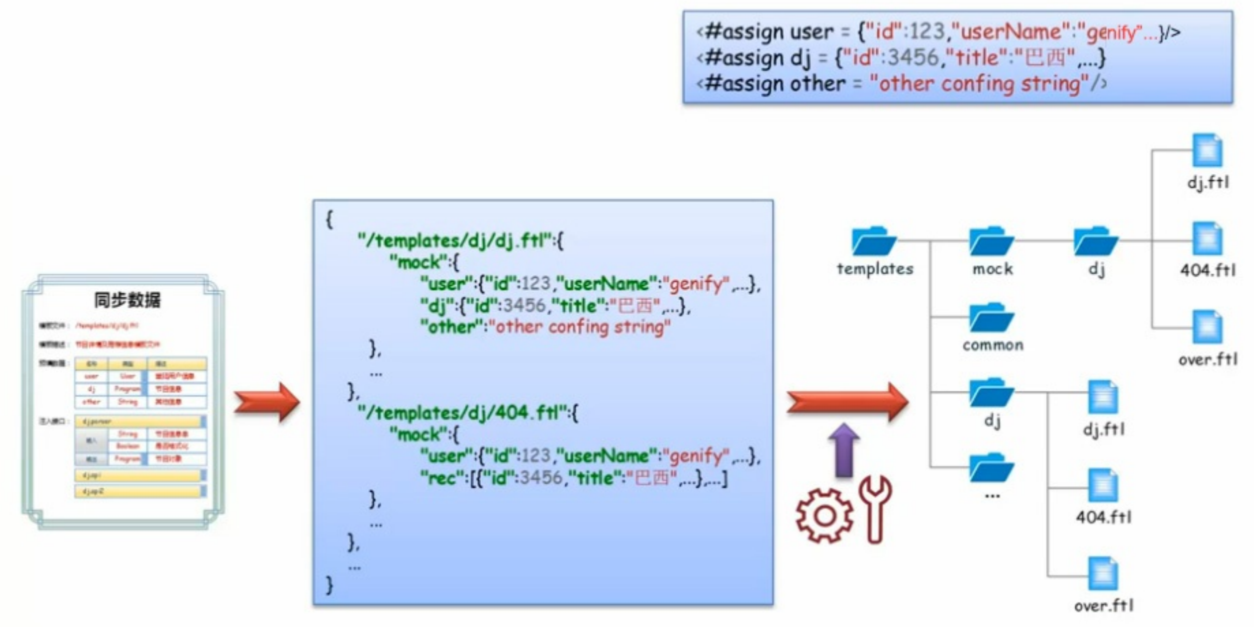
- 接口基本信息，地址不带查询参数
- 输入数据，REST，查询数据（必须包含名称类型及描述）
- 输出结果，通用部分及结果集（复杂的信息需展开说明）

规范的应用

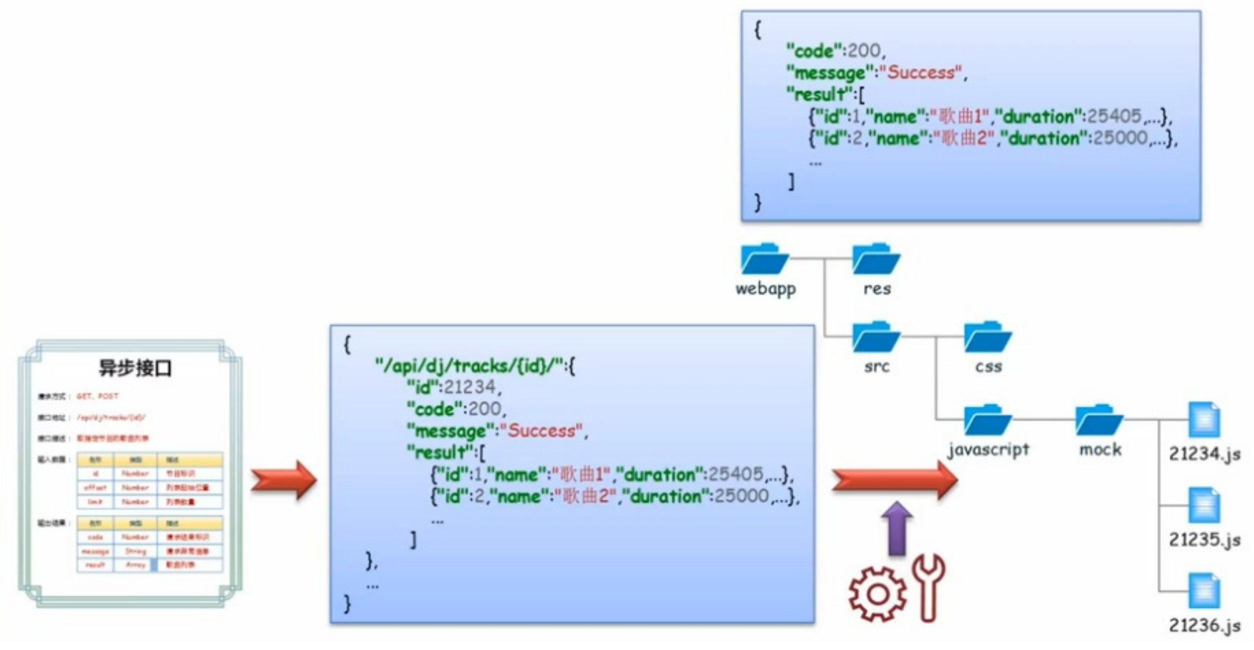
通过模拟数据的形成，将前端本地开发与后端独立出来， 这样前端工程师就可以独立的进行本地的开发工作。



使用页面入口规范制定项目结构（配置信息，目录结构和模板结构） ， 此过程可以使用自动化工具自动完成。



根据同步数据规范可生成模拟数据的配置文件。（此部分通用可以使用自动化工具来完成）



根据异步接口规范生成模拟异步数据。

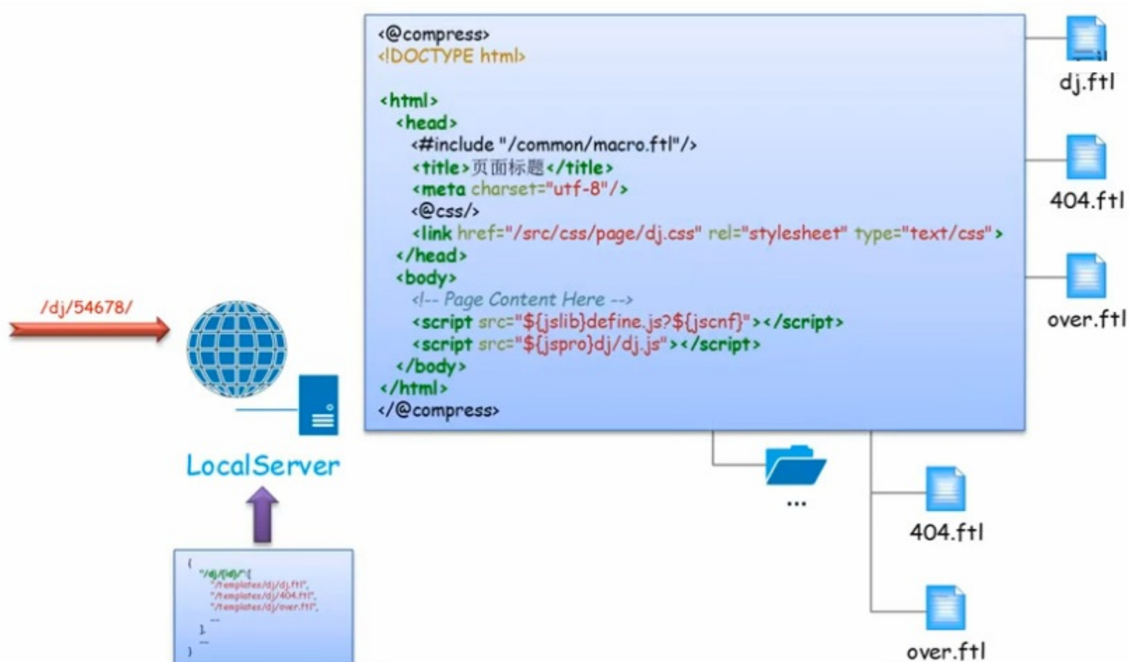
本地开发



前端开发环境包含两个部分，本地模拟服务器和本地代理。

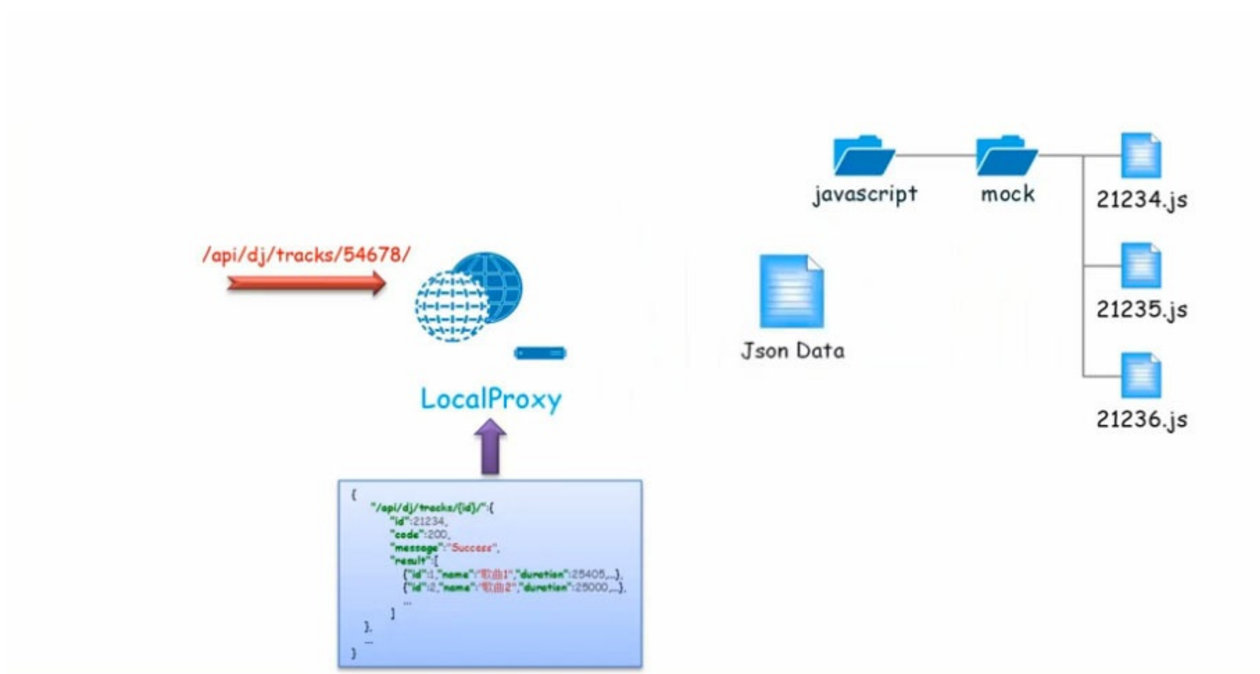
1. 客户端发送的请求，会被本地模拟服务器拦截并返回相应的模拟数据
2. 客户端发送的异步请求，会被本地代理拦截并返回对应的模拟数据

Local Server



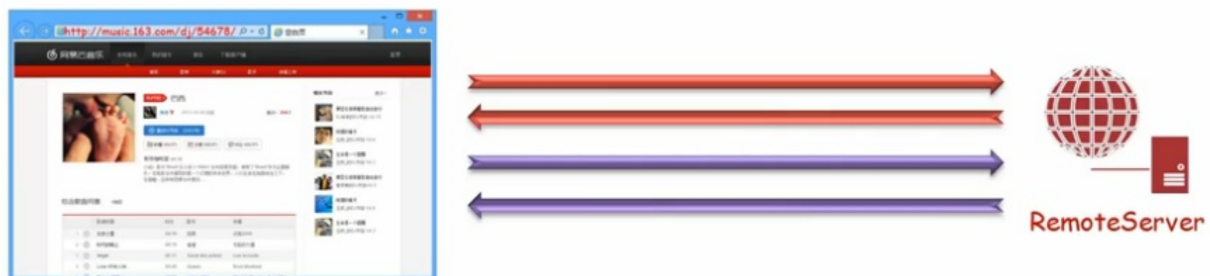
根据请求规则进行匹配，然后生成（整合模板和模拟数据）所请求的页面

Local Proxy



拦截异步请求后，根据请求的匹配规则返回所请求的数据（例如 JSON 或 XML）。

联调



前后端联调需要去除本地环境，在实际开发中只需要对配置文件进行调整既可（控制哪些请求需要被本地服务器或代理拦截，哪些需要使用远程服务器）。

版本控制

版本管理涉及团队协作，产品质量，和产品上线。使用版本控制工具可使我们自由的做的一些几点：

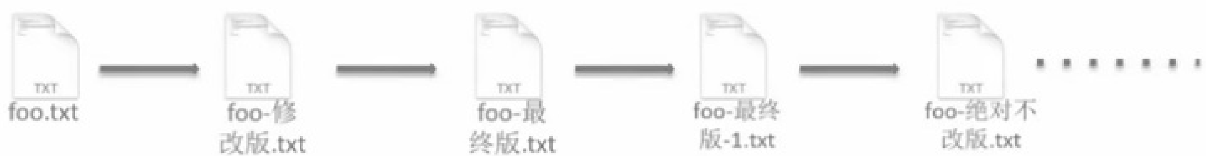
- 回退到任意版本
- 查看历史版本
- 对比两个版本差异

版本控制系统

版本控制系统（Version Control System）是一种记录若干文件修订记录的系统，它可以帮助开发者查阅或回档至某个历史版本。

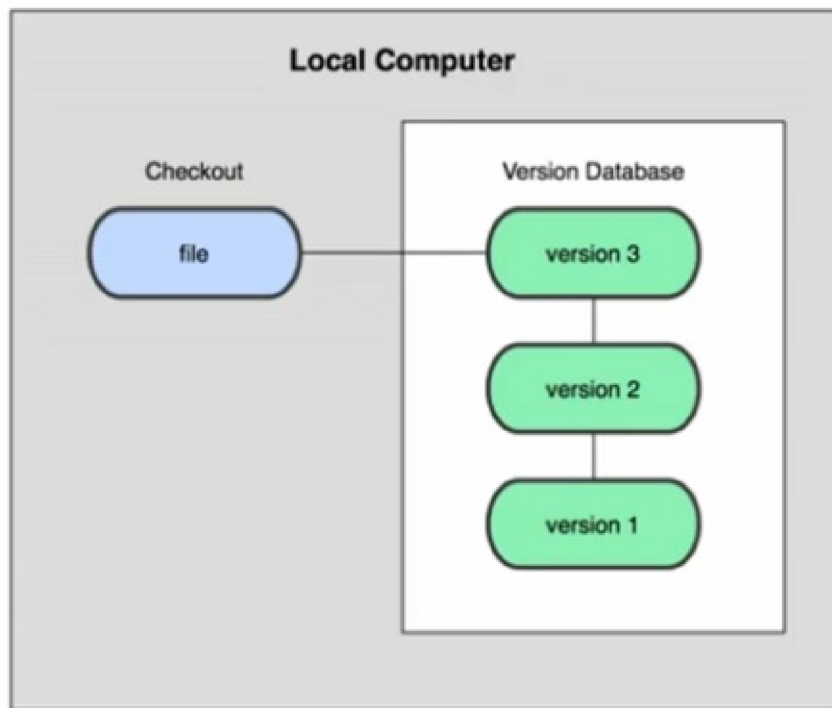
- 手动版本控制
- LVCS 本地
- CVCS 集中式（例如 SVN）
- DVCS 分布式（例如 Git）

手动版本控制



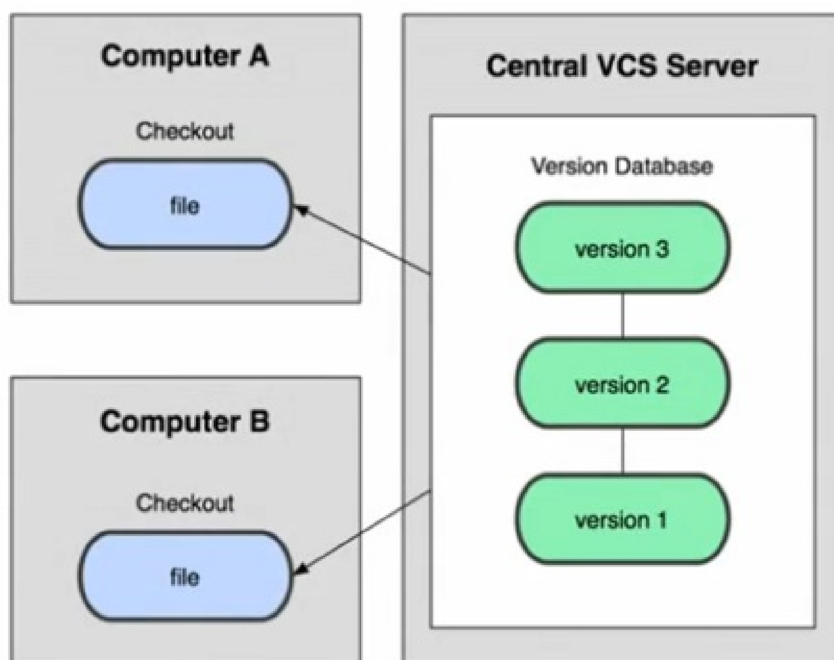
无法有效找到需要版本和差异，污染工作目录结构。

Local VCS 本地式



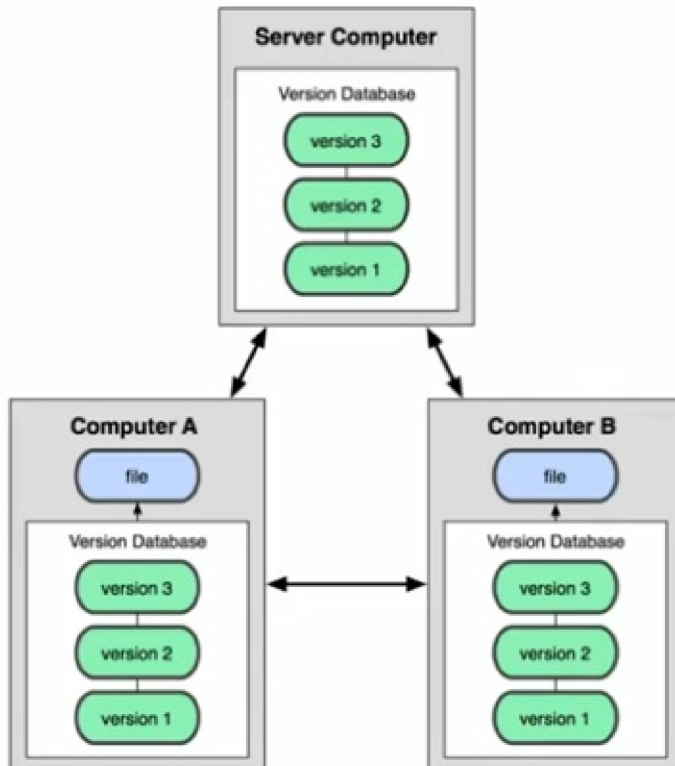
于是出现了本地版本控制系统 RCS（Reversion Control System），其利用本地版本数据库存储不断出现的文件版本。 它可以迅速找出需求的版本和维持工作目录结构。其缺点是不支持协同开发，这也让开发者不将其选做通用的版本控制工具来使用。

CVCS 集中式



利用中央服务器来管理文件版本，但每一次操作都需要网络请求，且具有致命的单点故障。（既中央服务器故障可导致，无法协同工作或数据丢失）

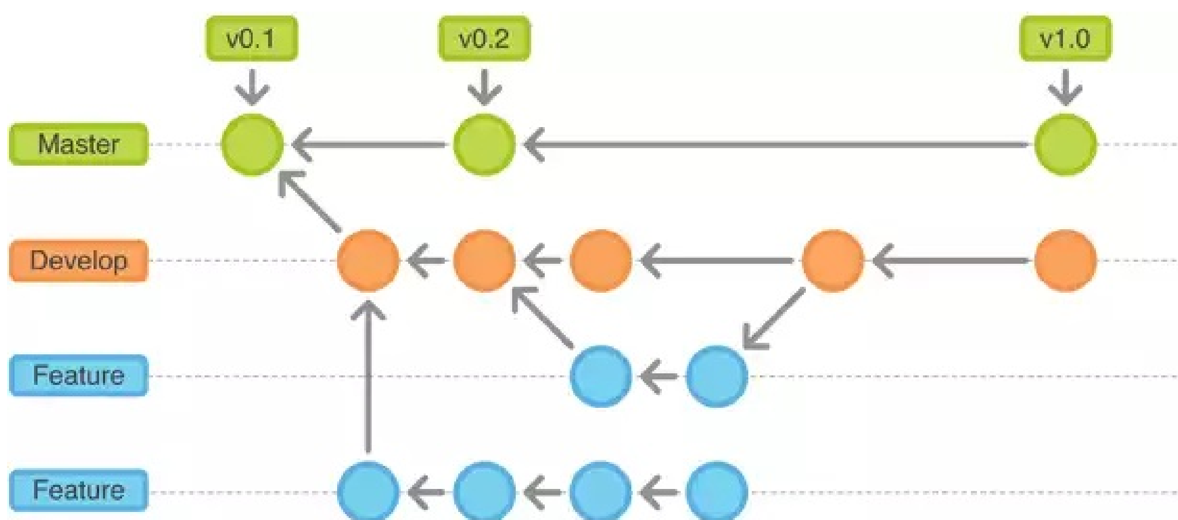
DVCS 分布式



分布式指的是每一份本地仓库都是一个完整的项目历史拷贝，即使一份仓库丢失或者损坏也可以从其他的仓库中获取此项目的完整历史记录。也因为在添加新版本不需要网络，这可以使操作流程。

分支模型

如果多人并行在一条线上开发会导致开发困难并且难以定位错误位置。



分支，就是从目标仓库获得一份项目拷贝，每条分支都具有和原仓库功能一样的开发线。

分支模型（Branching Model）或称之为工作流（Workflow），它是一个围绕项目开发、部署、测试等工作流的分支操作（创建及合并等）的规范集合。

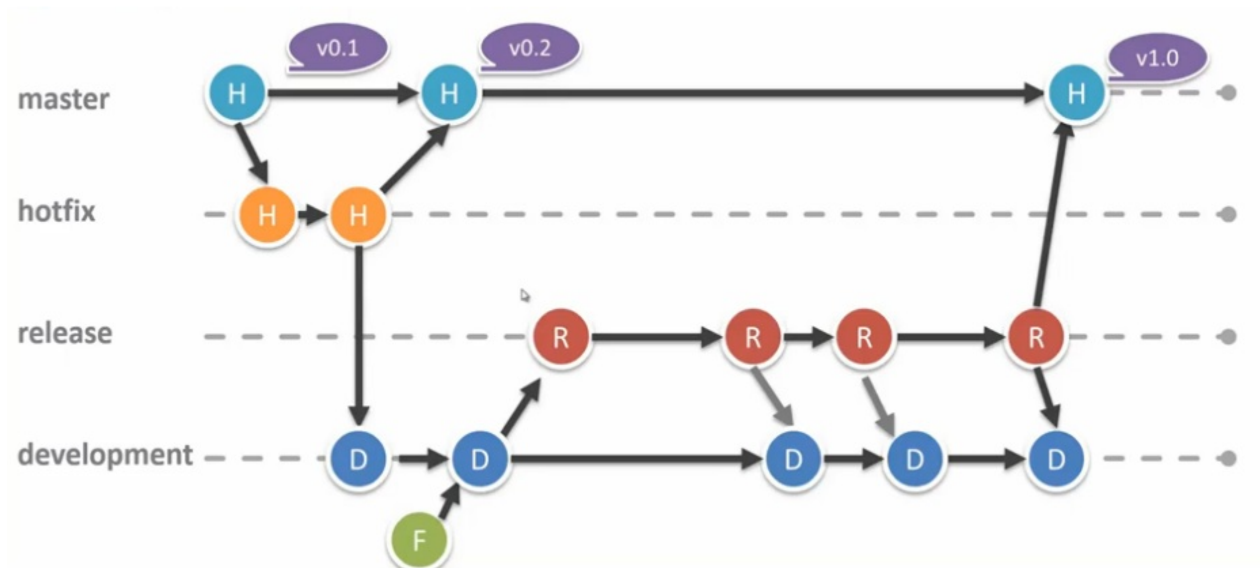
产品级开发分支模型

常驻分支

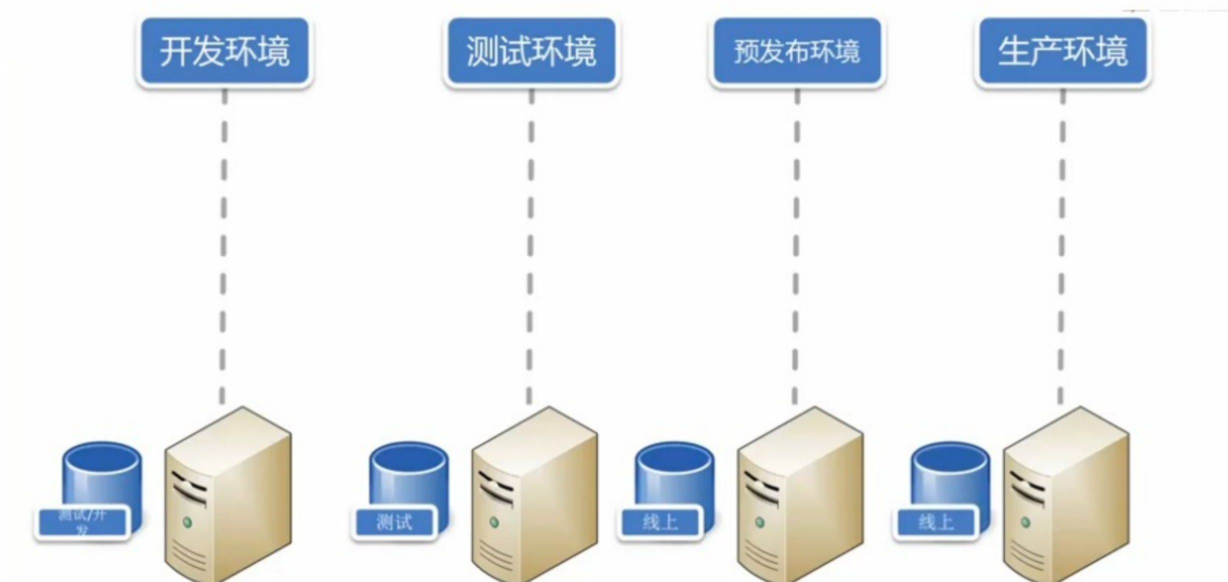
- development，从 master 创建
- production（master），默认分支可用于发布的代码

活动分支

- feature，从 development，创建其为特性分支
- hotfix，从 master 创建，用于修复 Bug
- release，从 development 创建



环境



- 开发环境，使用测试开发配置（数据库，缓存，元数据配置）
  - 使用提交到下一个 release 的特性分支
- 测试环境，使用测试配置（测试数据库）
  - 使用 release/development
- 预发布环境，小范围发布使用线上数据库模拟真实环境
  - 使用 release
- 生产环境，线上配置
  - 使用 master

Git

Git 是一个免费开源的分布式版本控制系统，它也一个基于内容寻址的存储系统。Git 是由 Linux 的创造者 Linus Torvalds。

优势

- 速度快，不依赖网络
- 完全分布式
- 轻量级分支操作
- Git 为行业标准版本控制供给
- 活跃和成熟的社区

安装

Mac OS X 下使用 `brew install git` 下载更新既可。

Linux Ubuntu 下可使用 `apt-get install git` 既可。

Git 介绍

Git 命令

常用 Git 命令，当在命令行中键入 `git` 可以便可以在帮助信息中看到。

```
 X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~  git
usage: git [--version] [--help] [-C <path>] [-c name=value]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

The most commonly used git commands are:
   add        Add file contents to the index
   bisect     Find by binary search the change that introduced a bug
   branch     List, create, or delete branches
   checkout   Checkout a branch or paths to the working tree
   clone      Clone a repository into a new directory
   commit     Record changes to the repository
   diff       Show changes between commits, commit and working tree, etc
   fetch      Download objects and refs from another repository
   grep       Print lines matching a pattern
   init       Create an empty Git repository or reinitialize an existing one
   log        Show commit logs
   merge      Join two or more development histories together
   mv         Move or rename a file, a directory, or a symlink
   pull       Fetch from and integrate with another repository or a local branch
   push       Update remote refs along with associated objects
   rebase     Forward-port local commits to the updated upstream head
   reset      Reset current HEAD to the specified state
   rm         Remove files from the working tree and from the index
   show       Show various types of objects
   status     Show the working tree status
   tag        Create, list, delete or verify a tag object signed with GPG

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
 ✘ X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~  |
```

获取帮助

`git help <command>` `git <command> -h` `git <command> --help` 还有 `man git-<command>` 均可查询某个命令的帮助文档。

基本操作

配置 Git 使用 `git config` 此为创建 Git 仓库前必须完成的配置。

```
git config --global user.name "Li Xinyang"
git config --global user.email "lixinyang1026@gmail.com"
```

- --local 默认 具有最高优先级 只影响本仓库 `.git/config`
- --global 中级优先级 影响到所有当前用户的仓库 `~/.gitconfig`
- --system 最低优先级 影响到全系统的仓库 `/etc/gitconfig`

初始化仓库

```
1. X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs: ~/Git-In-Action (zsh)
X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~/Git-In-Action  clear
X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~/Git-In-Action  git init
Initialized empty Git repository in /Users/X/Git-In-Action/.git/
X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~/Git-In-Action  master  tree .git
.git
├── HEAD
├── config
├── description
├── hooks
│   ├── applypatch-msg.sample
│   ├── commit-msg.sample
│   ├── post-update.sample
│   ├── pre-applypatch.sample
│   ├── pre-commit.sample
│   ├── pre-push.sample
│   ├── pre-rebase.sample
│   ├── prepare-commit-msg.sample
│   └── update.sample
├── info
│   └── exclude
├── objects
│   ├── info
│   └── pack
└── refs
    ├── heads
    └── tags
```

初始化仓库，使用 `git status` 可以查询当前仓库的状态。 如在未初始化仓库时查询状态会报出错误信息。

```
git init [path]
git init [path] --bare
```

在初始化仓库后会出现一个隐藏的目录 `.git` 其中包括了所有的当前仓库的版本信息和本地设置文件(`.git/config`)。

```
1. X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs: ~/Git-In-Action (zsh)
X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~/Git-In-Action  master  git status
On branch master

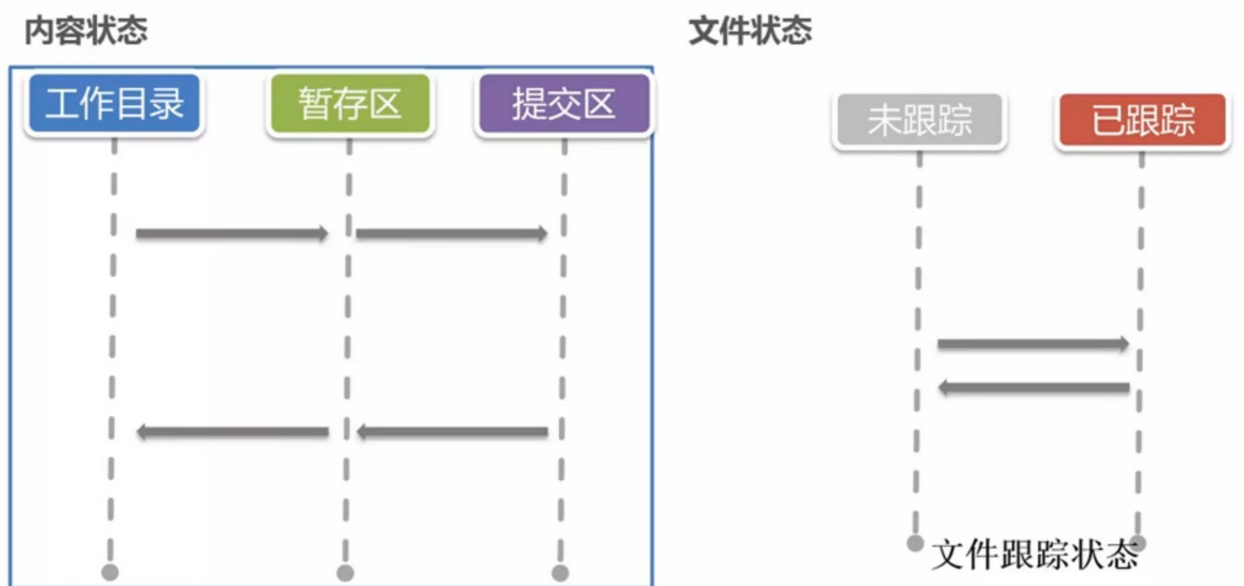
Initial commit

nothing to commit (create/copy files and use "git add" to track)
X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~/Git-In-Action  master
```

查询状态

`git status` 此命令可以帮助开发者在下面三对关系中找出文件状态的变化。

- 未跟踪 <--> 跟踪
- 工作目录 <--> 暂存区
- 暂存区 <--> 最新提交



Git 中存在两种状态内容状态和文件状态。 仓库中的文件均可以在状态和区域之间进行转换。

```
1. X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs: ~/Git-In-Action (zsh)
 X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~/Git-In-Action  master  touch README.md
 X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~/Git-In-Action  master  git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        README.md

nothing added to commit but untracked files present (use "git add" to track)
 X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~/Git-In-Action  master  |
```

添加文件到暂存区（同时跟踪文件）

```
git add [file]
```

```
1. X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs: ~/Git-In-Action (zsh)
 X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~/Git-In-Action  master  git add README.md
 X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~/Git-In-Action  master  git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   README.md

 X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~/Git-In-Action  master  |
```

上图所示，我们将 `README.md` 文件从工作区提交至暂存区，并将文件状态从未跟踪改变成已跟踪。



NOTE：批量增加当前目录下全部文件 `git add .`

```
1. X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs: ~/Git-In-Action (zsh)
X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~/Git-In-Action  master  git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        DummyFile0.txt
        DummyFile1.txt
        DummyFile2.txt

X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~/Git-In-Action  master  git add .
X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~/Git-In-Action  master  git status
On branch master

Initial commit

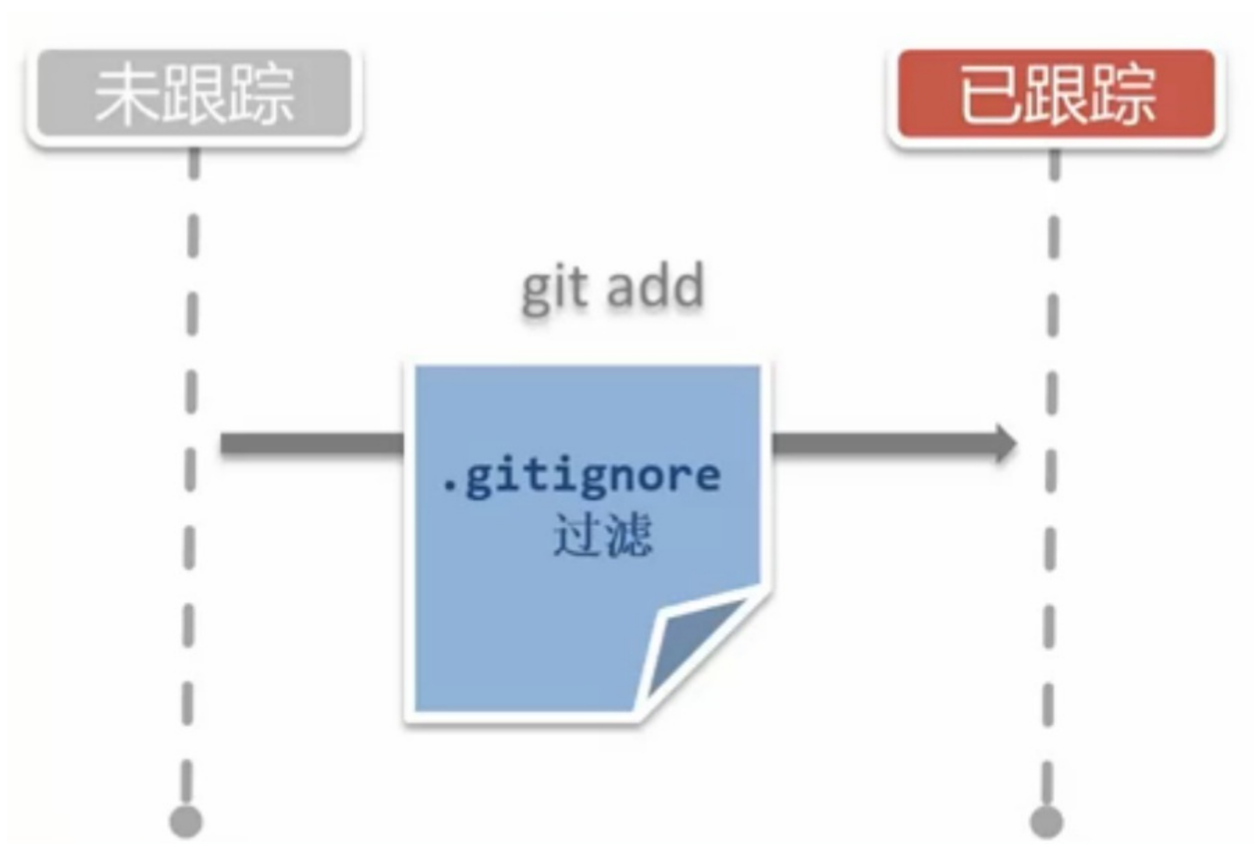
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   DummyFile0.txt
        new file:   DummyFile1.txt
        new file:   DummyFile2.txt
        new file:   README.md

X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~/Git-In-Action  master  |
```

忽略文件

`.gitignore` 可以在添加至仓库时忽略匹配的文件，但仅作用于未跟踪的文件。



NOTE：GitHub 为各个类型项目和操作系统提供了忽略文件模板，可以在这里找到。

暂存区删除文件

使用 `git rm` 可以做到从暂存区删除文件，下面提供几种常用的使用方法：

- `git rm --cached` 仅存暂存区删除
- `git rm` 才暂存区和工作区目录中删除
- `git rm $(git ls-files --deleted)` 删除所有被跟踪但在工作区已经被删除的文件

NOTE： `git-ls-files`

- Show information about files in the index and the working tree

工作区与暂存区

不同的区域中可以存在文件的独立版本，如下图所示工作区和暂存区的文件为两个不同的版本。（之前上个例子中所创建的 `DummyFile` 文件已被删除）

```
1. X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs: ~/Git-In-Action (zsh)
X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~/Git-In-Action  master  git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   README.md

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~/Git-In-Action  master 
```

暂存区

我们可以把暂存区比作一个每类物品只能放置一次的购物车此外还具有下面的特质：

- 货架和购物车可同时出现同种物品
- 货架上的物品可以替换掉购物车的物品
- 可以删除物品
- 提交购物车完成购买并生成购买记录

其中

- 物品：文件
- 货架：工作目录
- 购物车：暂存区
- 购买：提交内容

提交版本记录



`git commit` 可以根据暂存区的内容创建一个提交目录。

```
1. X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs: ~/Git-In-Action (zsh)
X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~/Git-In-Action  master  git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   README.md

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~/Git-In-Action  master  git commit -m 'Initial commit'
[master (root-commit) 7aaa5d4] Initial commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 README.md
X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~/Git-In-Action  master ●  git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~/Git-In-Action  master ●
```

NOTE：直接提交工作区的内容 `git commit -a -m 'message'` ，工作中不建议这样操作。

查询提交历史记录

`git log` 可以用来显示提交是记录信息。

```
commit 7aaa5d4cf30c04b6ec1b67eb11e5a9eba495761d
Author: Li Xinyang <lixinyang1026@gmail.com>
Date:   Fri Jul 3 11:18:23 2015 +0800

    Initial commit
(END)
```

其中包括：

- 提交编号 SHA-1 编码的 HASH 标示符
- git-config 配置的提交者信息
- 提交日期
- 提交描述信息

工作中可使用下面简单的配置进行版本查询

```
git log --oneline

# 较长的命令可以使用 alias 的方法简化
git log --color --graph --pretty=format:'%Cred%h%Creset -%C(yell
ow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev
-commit
```

Git 中 alias 命令设置

配置 Git 中别名的方法 `git config alias.shortname <fullcommand>` 。

```
git config --global alias.lg "log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit"

git lg
```

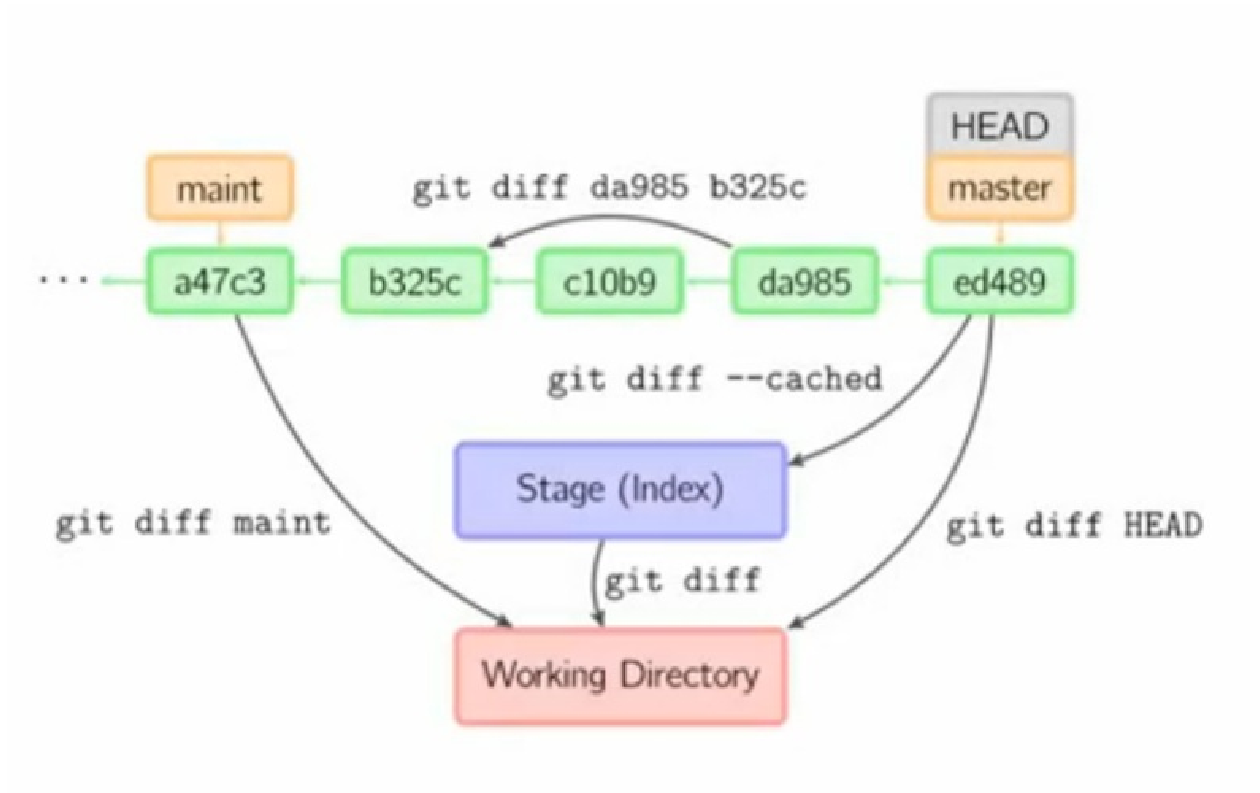
NOTE：如果你使用 Mac OS X 可以尝试使用 Oh My Zsh 其中已经预先设置好了非常多常用别名。

显示版本差异

```
1. git diff (git)
diff --git a/README.md b/README.md
index e69de29..8ab686e 100644
--- a/README.md
+++ b/README.md
@@ -0,0 +1 @@
+Hello, World!
(END)
```

`git diff` 用于显示版本差异，下面是几个常用的方法：

- `git diff` 显示工作目录与暂存区的差异
- `git diff -cached [<reference>]` 暂存区与某次提交的差异（默认为 HEAD）
- `git diff <reference>` 工作目录和某次提交间的差异
- `git diff <reference> <reference>` 查询两次提交直接的差别


HEAD
maint
git diff da985 b325c
master
...
a47c3
b325c
c10b9
da985
ed489
git diff --cached
Stage (Index)
git diff maint
git diff HEAD
git diff
Working Directory

撤销工作区的修改



`git checkout -- <file>...` 可用于撤销工作区的修改（此方法会丢弃工作区修改且不可恢复），下面是一些常用方法：

- `git checkout -- <file>` 将文件从暂存区复制到工作目录

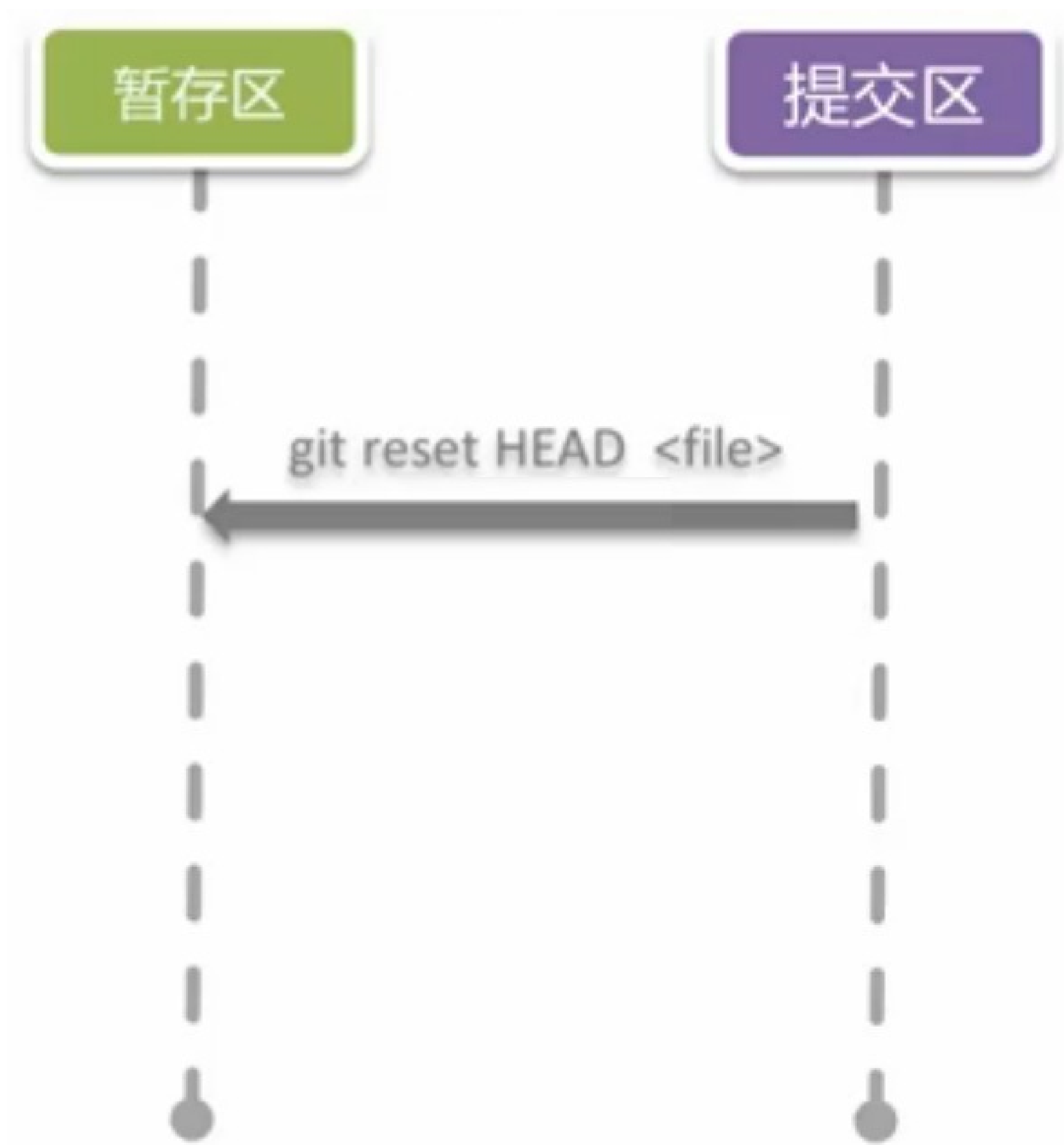
```
1. X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs: ~/Git-In-Action (zsh)
X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~/Git-In-Action  master ●  git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~/Git-In-Action  master ●  git checkout -- README.md
X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~/Git-In-Action  master  git status
On branch master
nothing to commit, working directory clean
X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~/Git-In-Action  master  |
```

NOTE：使用 `--` 是为了避免路径和引用（或提交 ID）同名发生的冲突。

撤销暂存区内容



使用 `git reset HEAD <file>...` 可用于撤销暂存区的修改，下面是一些常用操作：

- `git reset HEAD <file>` 将文件内容存上次提交复制到暂存区。

```
1. X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs: ~/Git-In-Action (zsh)
X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~/Git-In-Action  master ✚  git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.md

X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~/Git-In-Action  master ✚  git reset HEAD README.md
Unstaged changes after reset:
M       README.md
X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~/Git-In-Action  master ●  git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~/Git-In-Action  master ●  |
```

撤销全部修改



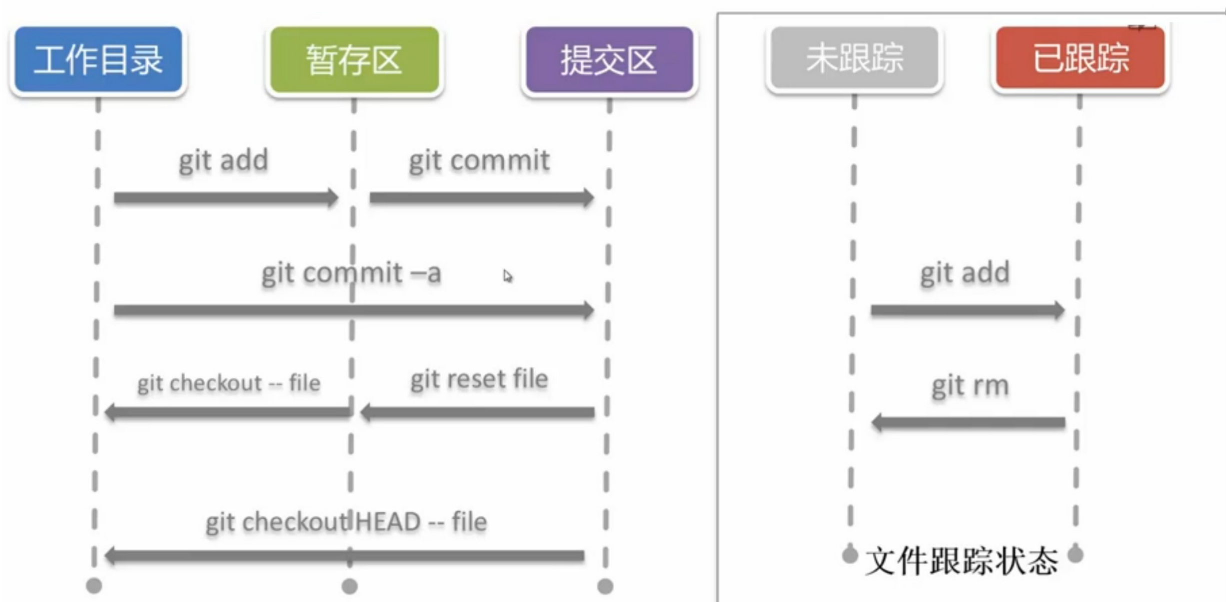
`git checkout HEAD -- <file>` 可以直接将内容从上次的提交复制到工作区。

```
1. X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs: ~/Git-In-Action (zsh)
X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~/Git-In-Action  master ✚  git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.md

X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~/Git-In-Action  master ✚  git checkout HEAD -- README.md
X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~/Git-In-Action  master  git status
On branch master
nothing to commit, working directory clean
X@Z9hbffLvnCyq8YQBnGnkgnwFLKUkUs  ~/Git-In-Action  master  |
```

命令总结



分支操作

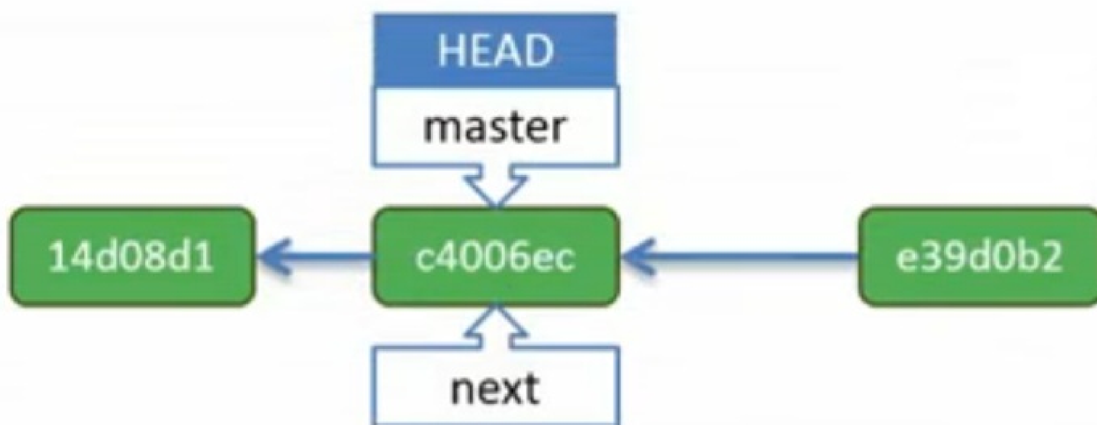
git branch

使用 `git branch` 可以对仓库分支进行增删查改的操作，下面列举了一下常用的操作方式：

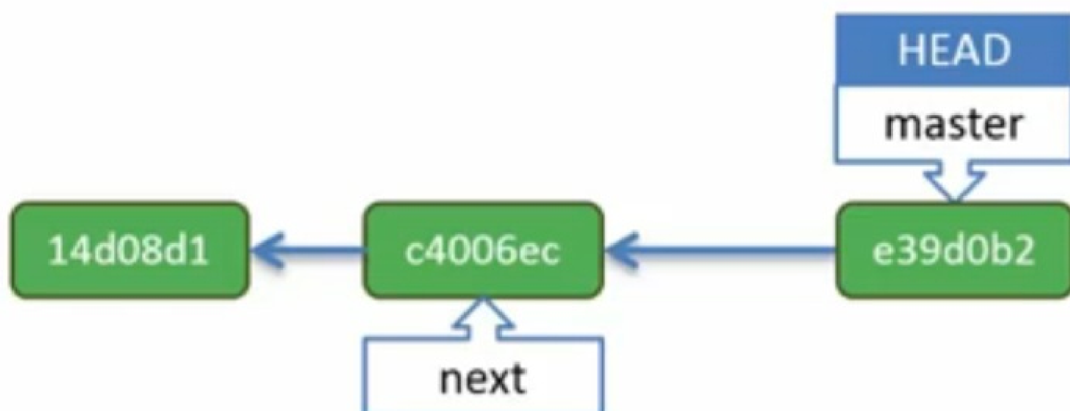
- `git branch <branchname>` ，创建指定分支
- `git branch -d <branchname>` ，删除指定分支
- `git branch -v` ，显示所有分支信息

> 一份分支的引用只是一个文本文件，里面只有一个 SHA 编码。它保存于 `.get/refs/heads/master` 中。
>
> —— 郑海波 网易工程师

```
git branch next
```




```
git commit -m 'message'
```

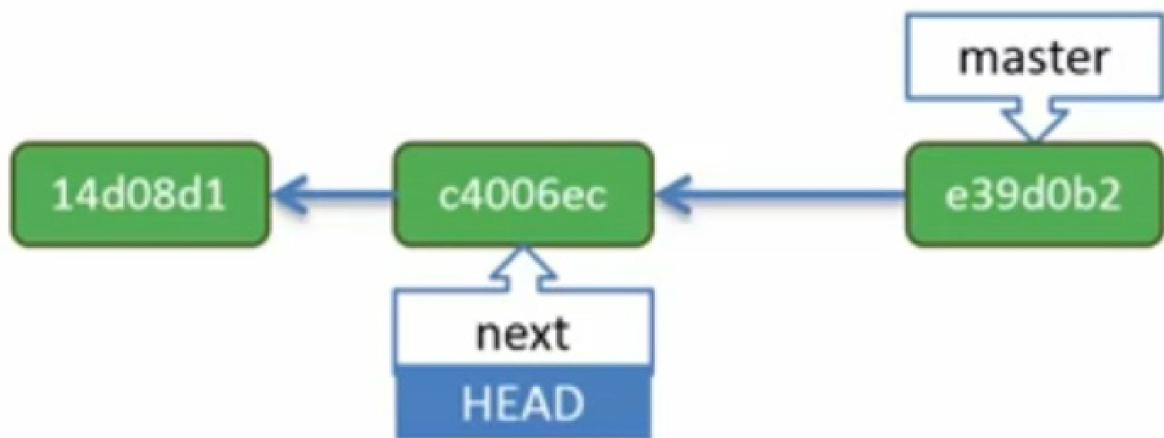


切换至目标分支

`git checkout` 它可以本枝上根据通过移动 HEAD（指向当前的提交） 检测出版本， 也可用于切换分支。（其会把当前的工作目录和暂存区移动到提出分支的版本） 常用命令有：

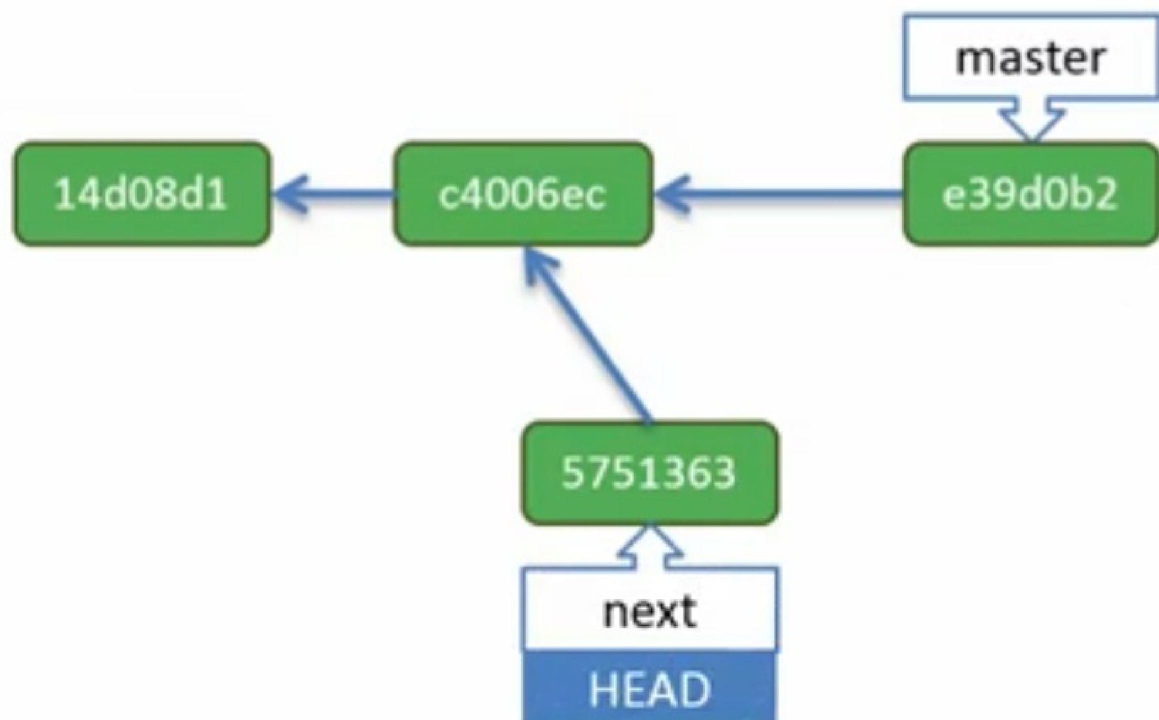
- `git checkout <branchname>` ，使指针指向目标分支
- `git checkout -b <branchname>` ，创建目标分支并切换分支
- `git checkout <reference>` ，可以指向任何一个版本

```
git checkout next
```



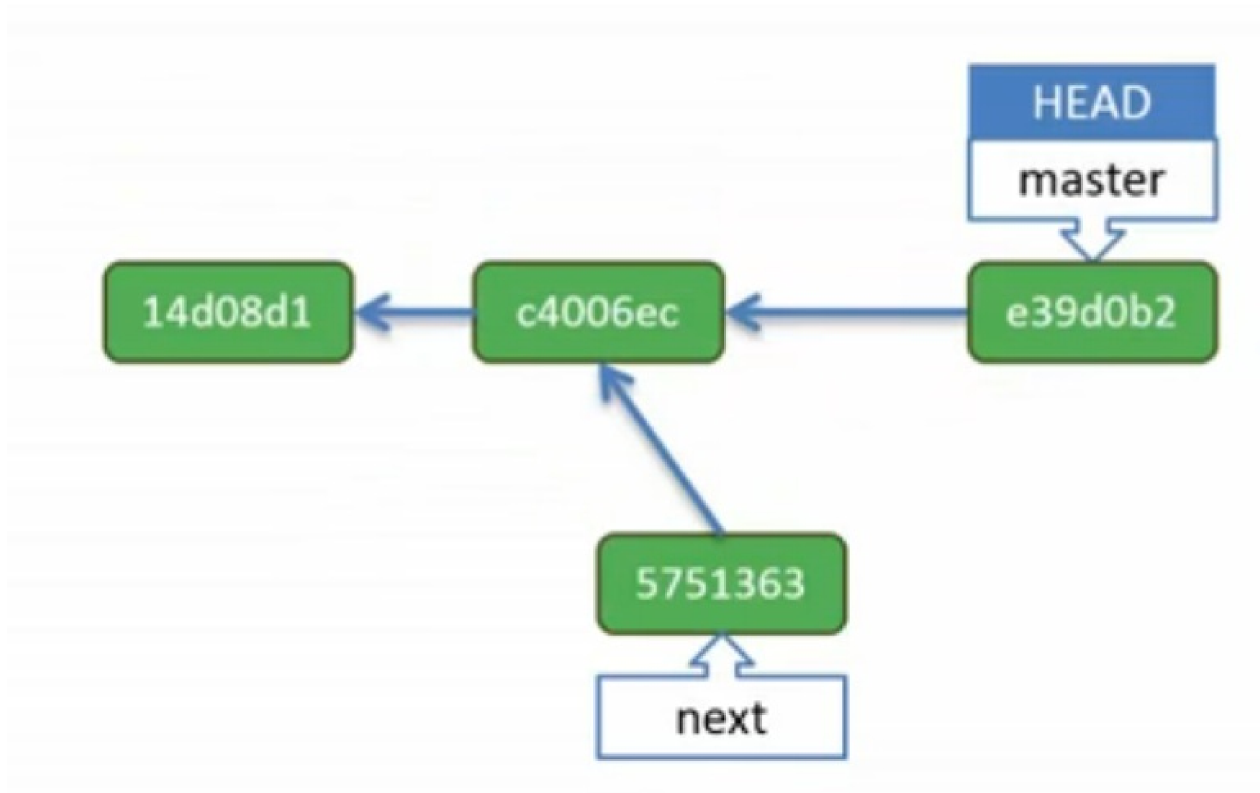
NOTE：所有提交是更具 HEAD 向前进的，所以前后分支后则会跟着 Next 分支进行开发。

```
git commit -m 'message'
```

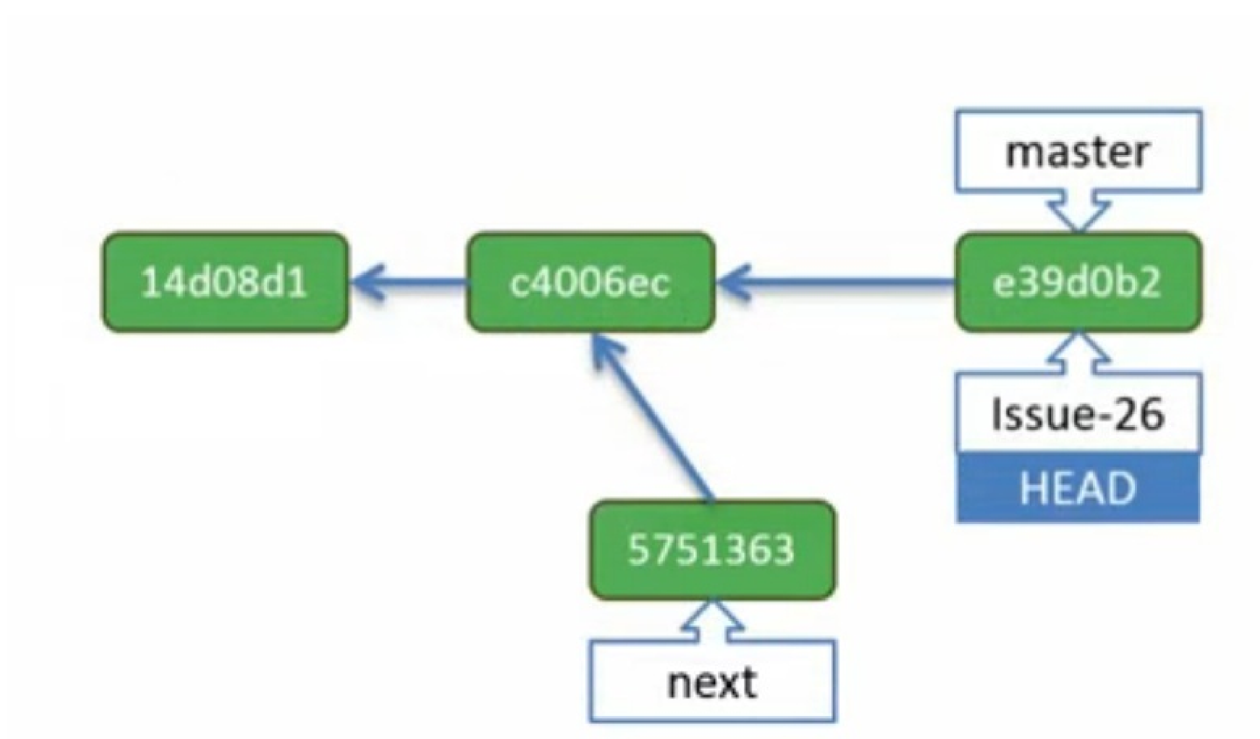



```
# -- 为短名与 cd 类似
git checkout --

# 或者使用
# git checkout master
```

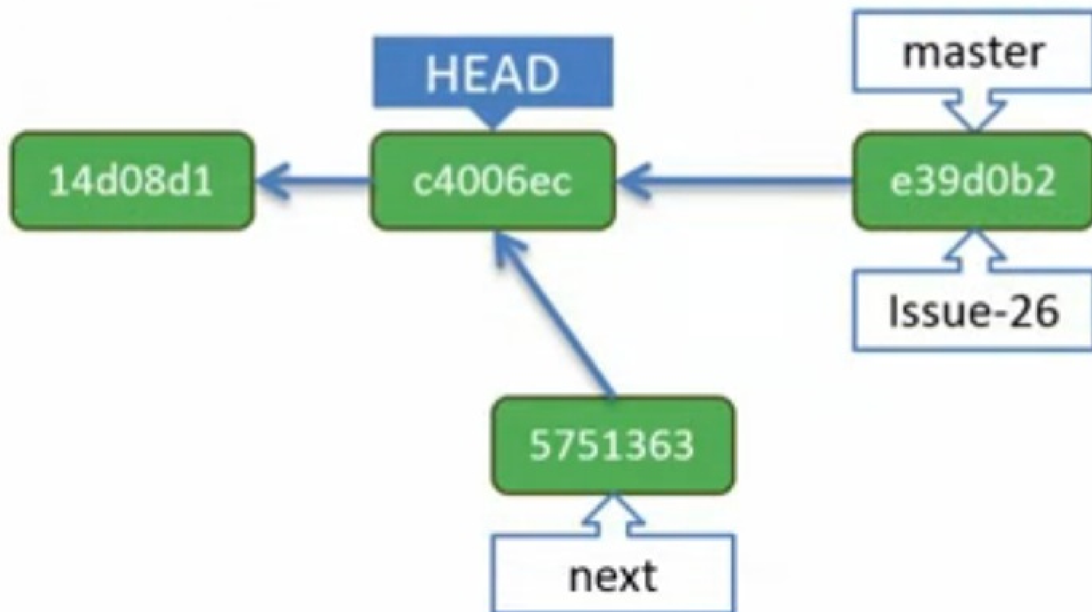



```
git checkout -b Issue-26
```

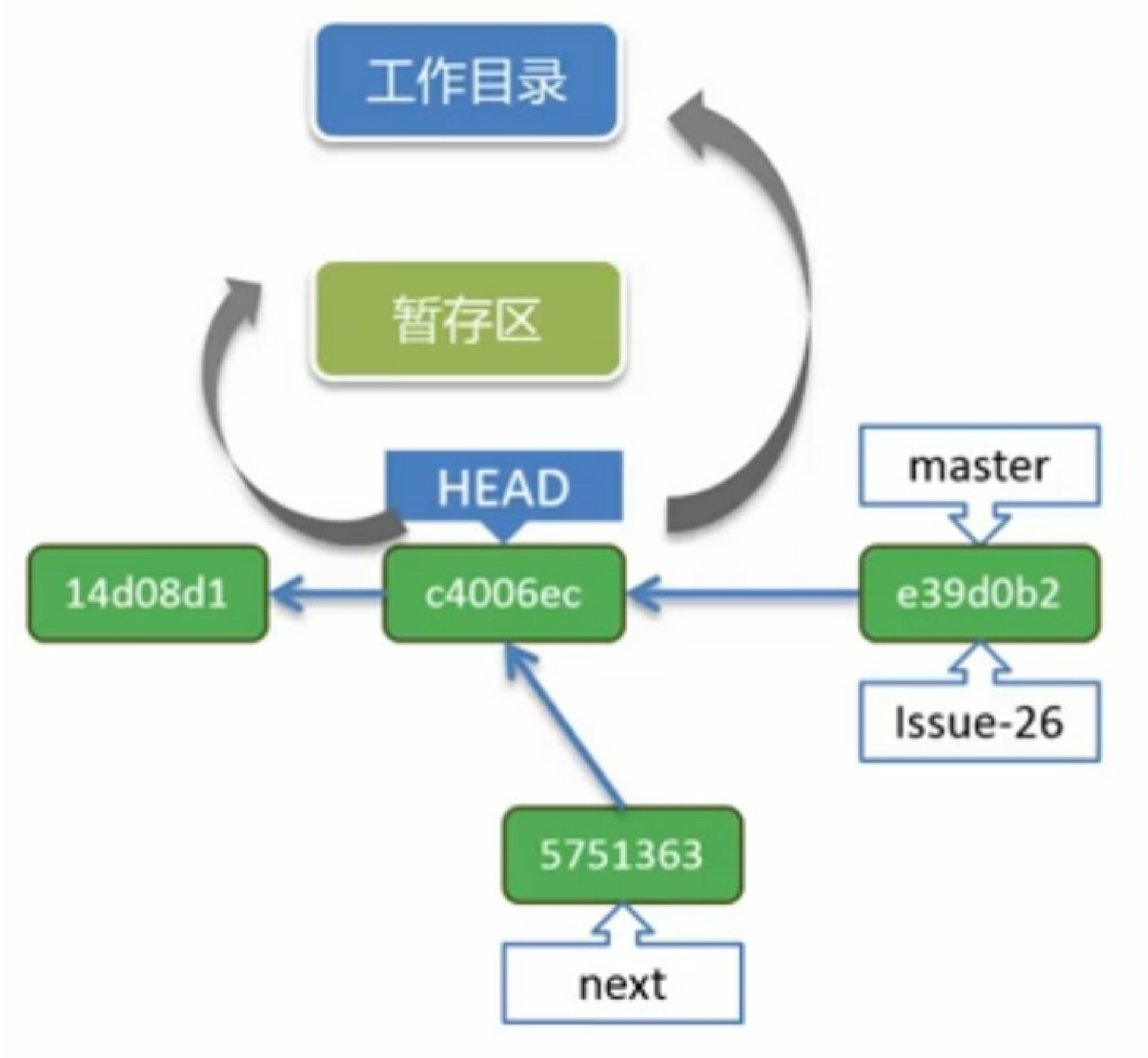


NOTE：使用 `git branch -v` 可以列出全部分支，带 `*` 表示当前所属分支（HEAD 指向分支）。

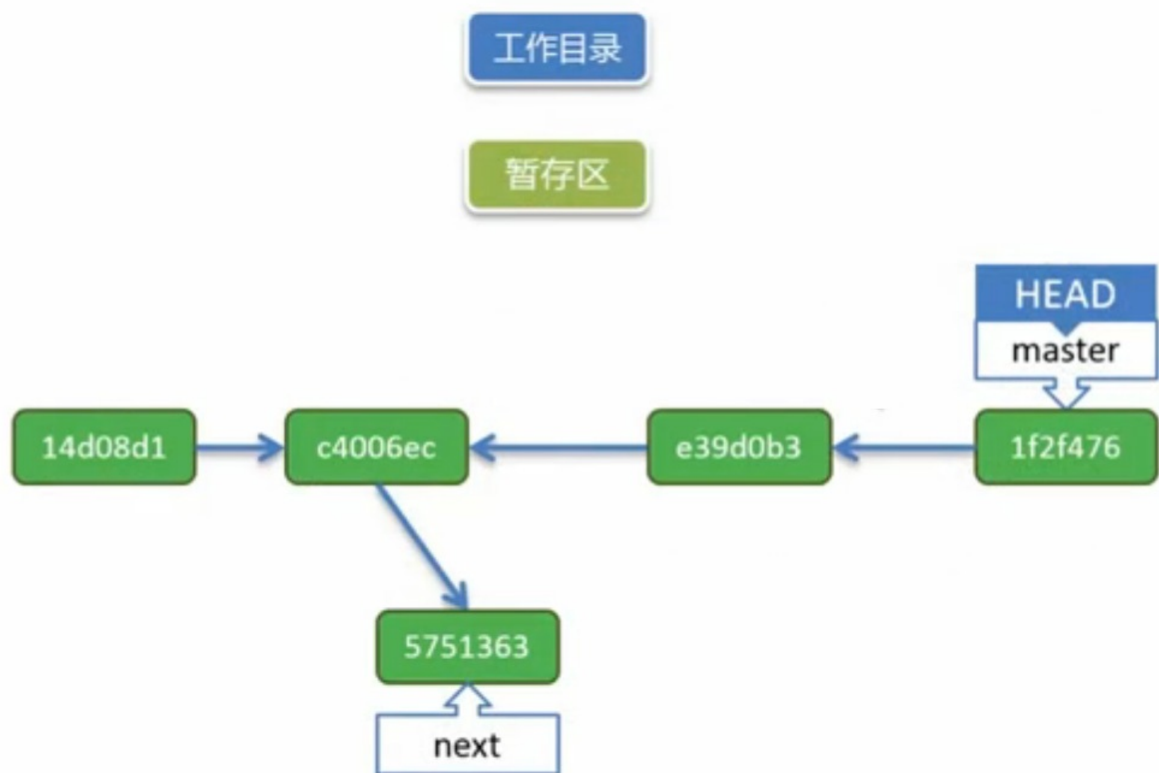
```
git checkout c4006ec
```



当 HEAD 指针与具体的分支分离时，我们将其称之为 `detached head` 。如果 HEAD 在分离状态则因尽量避免在此状态下进行提交，只做内容的查看。


工作目录
暂存区
HEAD
master
14d08d1
c4006ec
e39d0b2
Issue-26
5751363
next

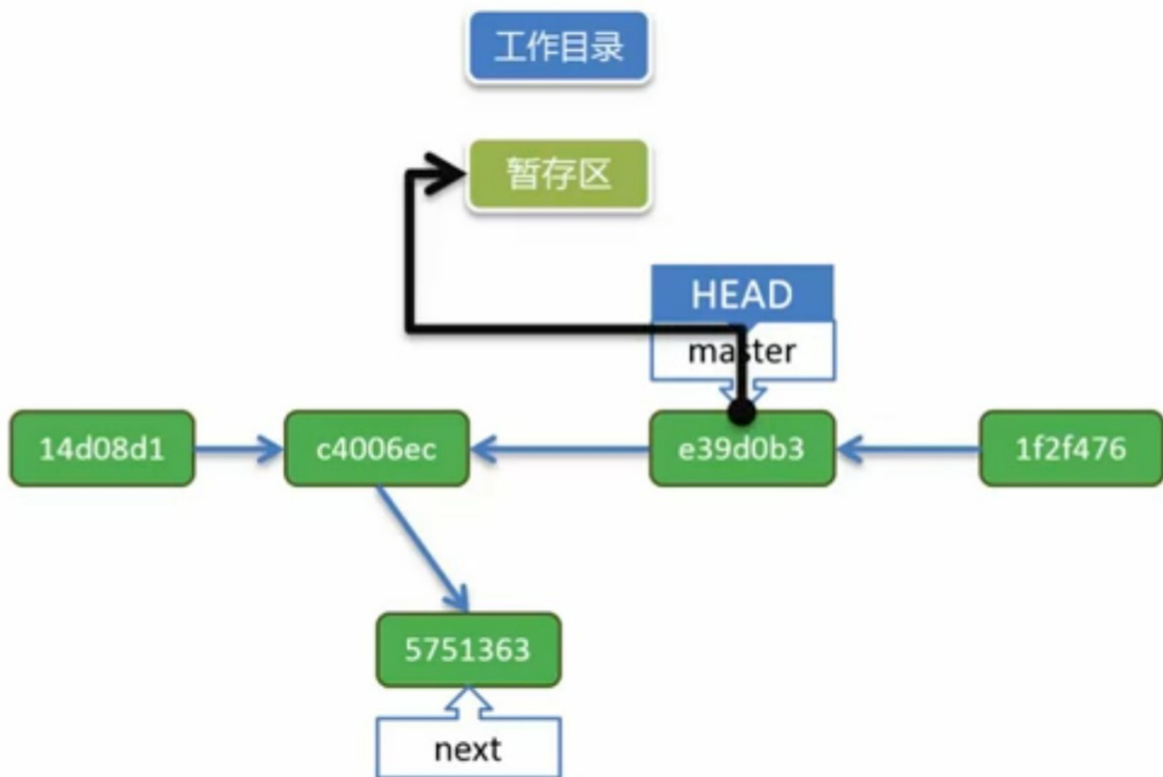
完全回退



使用 `git reset` 可以将当前分支回退到历史中的某个版本，下面为常用的三种方式（三种的区别是恢复的内容时候同时会恢复的工作区或暂存区）：

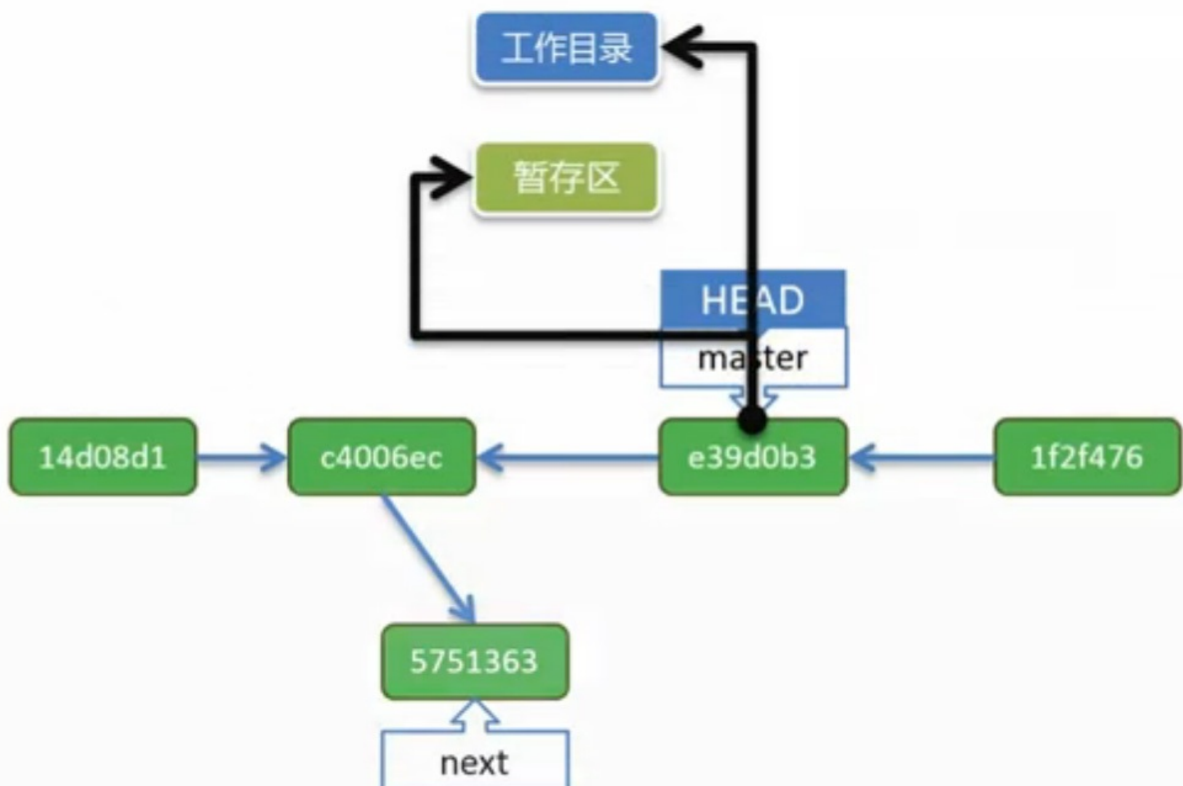
- `git reset --mixed <commit>` 默认方式，内容存入暂存区
- `git reset --soft <commit>` 内容存入暂存区和工作区
- `git reset --hard <commit>` 暂存区和工作区保留现有状态

```
git reset --mixed e390b3
```



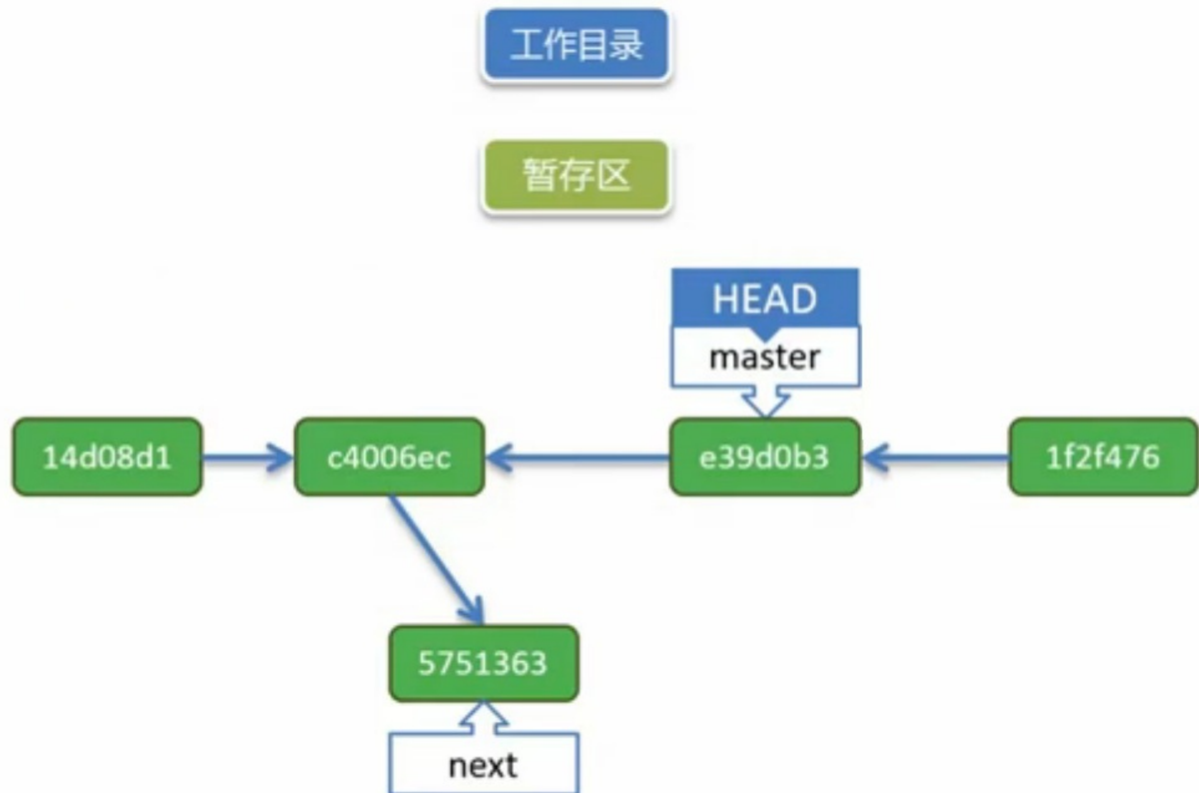
如果上一个命令如果使用 **hard**

```
git reset --hard e390b3
```



如果上一个命令如果使用 **hard**

```
get reset --soft e390b3
```



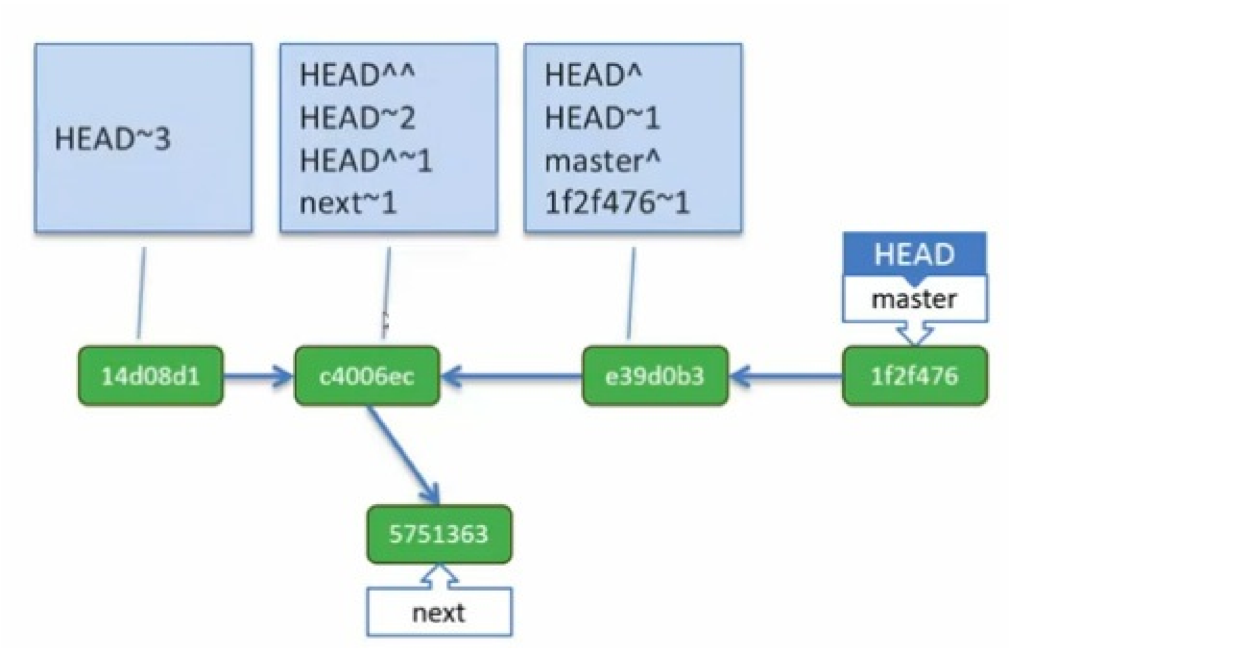
此方法暂存区和工作目录不会发生任何变化仅仅只是 HEAD 指针发生了变化， 但原有的提交已经无指针指向成为无索引的提交其就有可能被回收。

查询所有提交记录

`git reflog` 会根据仓库的提交顺序按顺序来排列，其中包括无索引的提交， 可以在这里使用 HASH 值来进行，但是无索引的提交可能会丢失。

使用捷径

`A^` 表示 `A` 上的父提交，多个 `^` 可表示以上的多个级别。 `A~n` 则表示在 `A` 之前的第 n 次提交。



reset 与 checkout 区别

两种方法都有两个作用范围，一个是分支操作（commit 操作），另一个是文件操作（file 操作）。

命令	范例	移动 HEAD/Branch	注释
`git reset [commit]`	`git reset HEAD^ --soft`	是/是	完全回退到某个提交（之前所在的位置将失去索）
`git reset [file]`	`git reset README.md`	否/否	恢复暂存区到某个提交状态（不移动指针）
`git checkout [commit]`	`git checkout master`	是/否	移动当前指针 HEAD 到某个提交（并复制内容到工作目录）
`git checkout [file]`	`git checkout -- README.md` `git checkout HEAD -- xx.log`	否/否	恢复工作目录到某个状态

stash 的作用

```
git checkout next

# error: Your local change to the following files whould be over
written by checkout:
#       README.md
# Please, commit your changes or stash them before you can switc
h branches.
# Aborting...
```

突然需要切换到其他分支，工作区和暂存区还有在当前分支没完成的任务。那么 `stash` 就使用 `.git` 中的特殊区（Stash 区）来帮你解决这个问题（因为强切回丢失当前的工作区和暂存区的内容）。



```
1. X@TimeMachine: ~/Desktop/Git-In-Action (zsh)
X@TimeMachine  ~/Desktop/Git-In-Action  master ●  clear
X@TimeMachine  ~/Desktop/Git-In-Action  master ●  git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
X@TimeMachine  ~/Desktop/Git-In-Action  master ●  git stash save 'push to stash'
Saved working directory and index state On master: push to stash
HEAD is now at 5247179 initial commit
X@TimeMachine  ~/Desktop/Git-In-Action  master  git status
On branch master
nothing to commit, working directory clean
X@TimeMachine  ~/Desktop/Git-In-Action  master  git stash apply stash@{0}
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
X@TimeMachine  ~/Desktop/Git-In-Action  master ●  git stash drop stash@{0}
Dropped stash@{0} (ef3fde8cc4ff4cac54c54fda33f2f55f28f664d4)
X@TimeMachine  ~/Desktop/Git-In-Action  master ●  git stash list
X@TimeMachine  ~/Desktop/Git-In-Action  master ●
```

`stash` 可以把当前工作区和暂存区的状态以栈（Stack）的形式保存起来（每次保存都会推一个内容到 `stash` 栈中），并返回一个干净的工作空间（工作区和暂存区）。

NOTE： `stash pop = stash apply + stash drop` 类似于 JavaScript 中的 `pop` 操作。

NOTE+：什么是栈？可以把栈想象成一摞盘子堆（一个叠一个）。具体关于堆栈的信息可以在这里找到。如果还看不懂，建议完成哈佛大学在线计算机入门课程 `CS50` ，可以在这里找到。

merge

使用 `git merge` 可用于合并分支。下面的例子是将 `next` 分支合并到 `master` 分支中去。

```
X@TimeMachine  ~/Desktop/Git-In-Action  master  git status
On branch master
nothing to commit, working directory clean
X@TimeMachine  ~/Desktop/Git-In-Action  master  git merge next
Auto-merging README.md
Merge made by the 'recursive' strategy.
 README.md | 2 ++
 1 file changed, 2 insertions(+)
X@TimeMachine  ~/Desktop/Git-In-Action  master  git status
On branch master
nothing to commit, working directory clean
X@TimeMachine  ~/Desktop/Git-In-Action  master  git cat-file -p HEAD
tree 0ea721be06feb39a8e579f3b6d0ddbc11a8dffb1
parent 86f58b5e65d24b0c0f1b8cbc120a0620dcea0bde
parent 49f8d85f4ad6f82bf467ac66245d0f16b19d7e04
author Li Xinyang <lixinyang1026@gmail.com> 1436243265 +0800
committer Li Xinyang <lixinyang1026@gmail.com> 1436243265 +0800

Merge branch 'next'
X@TimeMachine  ~/Desktop/Git-In-Action  master  |
```

解决 merge 冲突

当一个文件被同时修改时（更多情况为同时修改相同的一行代码时）则极有可能产生合并冲突。

```
git merge next master
# Autom-merging README.md
# CONFLICT (content): Merge confilict in README.md
# Automatic merge failed; fix confilict and then commit the result

git status
# On branch master
# You have unmerged paths.
#   (fix confilict and run 'git commit')
# ...
#   both:modified: README.md
# no changes added to commit (use "git add" and/or "git commit -a")
```

```
<<<<<<< HEAD
  $bind: function(component, expr1, expr2){
    var type = _.typeOf(expr1);

=======
  bind: function(component, expr1, expr2){
    var type = typeof expr1

>>>>>>> origin/master
    if( expr1.type === 'expression' || type === 'string' ){
```

在解决完合并冲突后可以使用 `git add .` 然后 `git commit -m 'resove merge confilict'` 来完成合并冲突解决并提交一个新的版本。

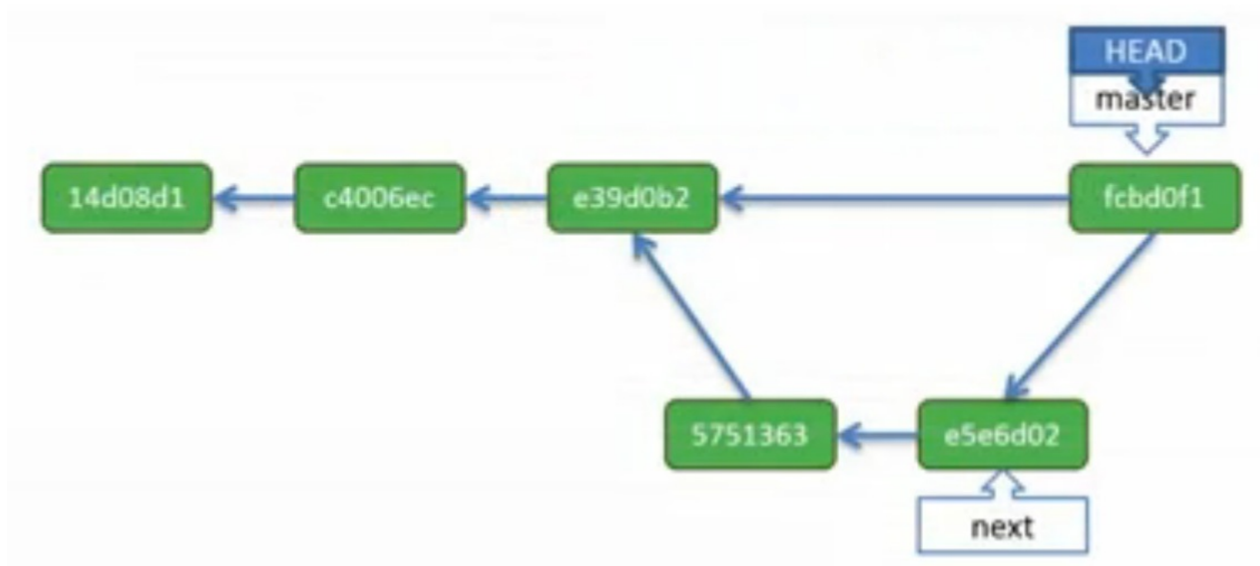
NOTE： `git cat-file -p HEAD` 可用于显示 `git` 中某个对象的具体信息。
NOTE+： `<<<<<<<< HEAD` 与 `=========` 之间为 HEAD 所在的内容。

merge fast-forward

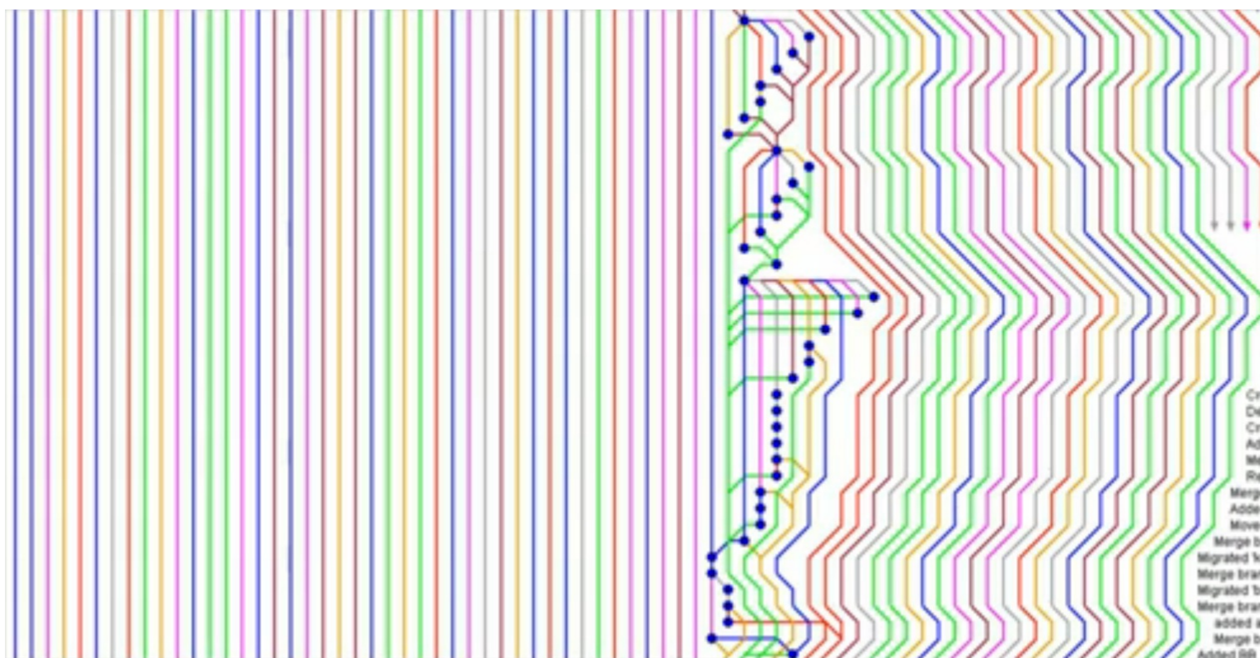
也并不是所有的合并操作都会造成合并从图（merge confilic)。最简单的一种合并是 `fast-forward` 仅仅只是变化 HEAD 指向的位置（不产生新的合并节点）。



如果需要生成新的合并节点可以使用 `git merge next --no-ff` 意思是合并但不使用 `fast-forward` 。



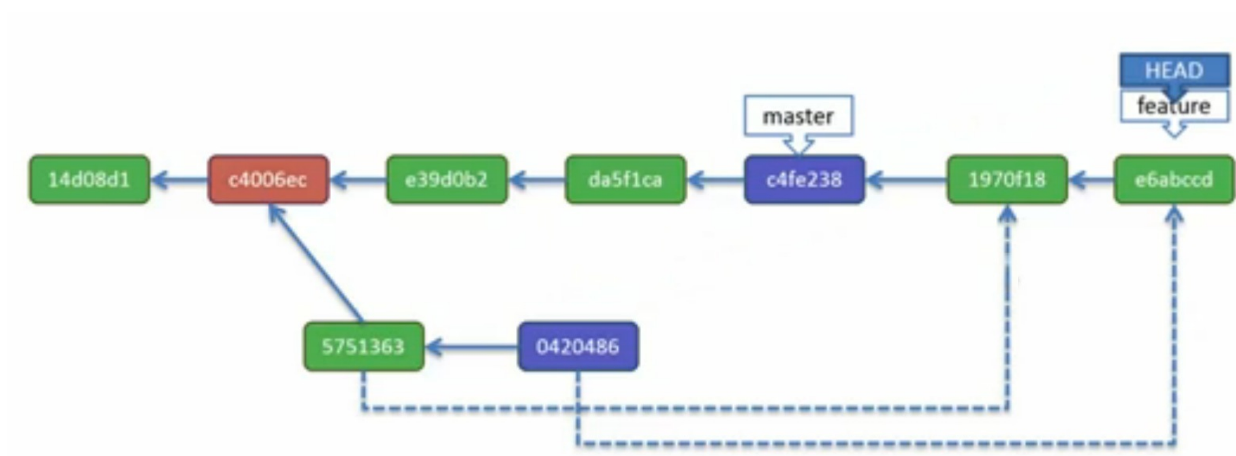
merge 不足



当参与的人阅读分支越多其分支结构就越复杂和难以被理解。如何实现在任何状态下的线性提交？

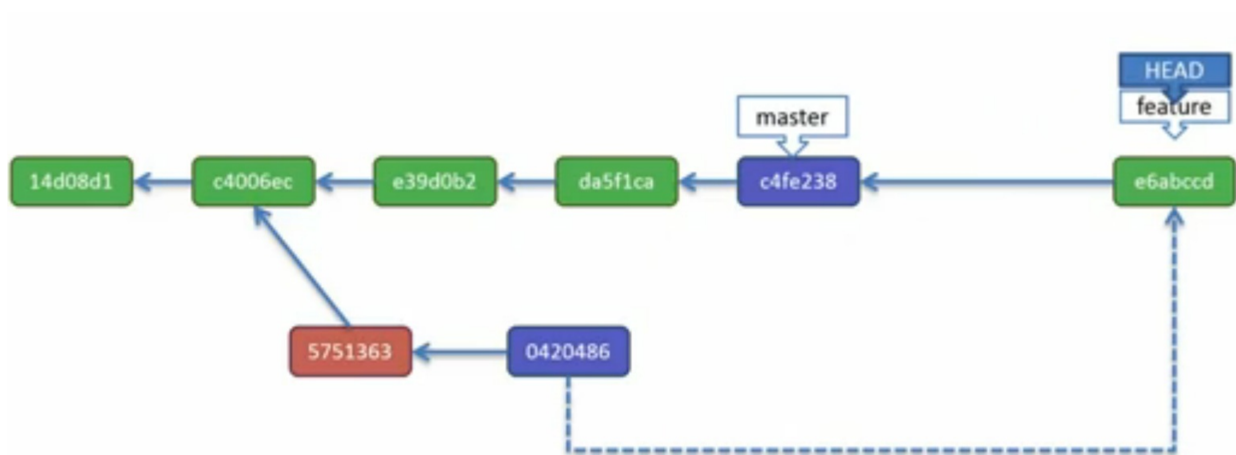
如需完成线性提交可以使用 `git rebase` ，其可以修剪提交历史的基线。它会将不同分支的提交在所选节点上进行重演（重演并重新创造新节点）这里 HEAD/Branch 均会发生移动。

```
git rebase master
```



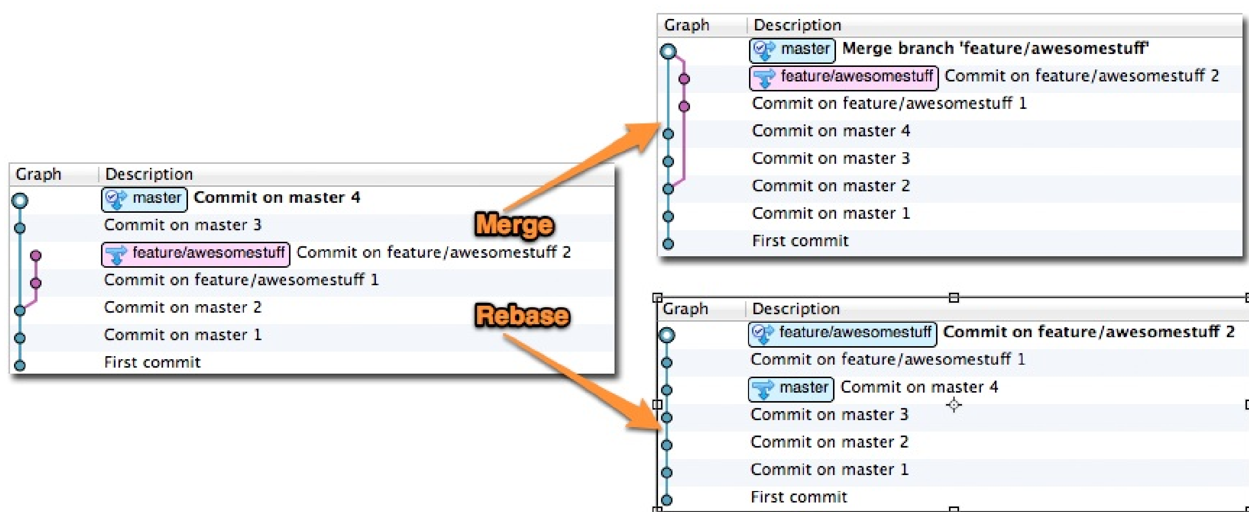
但有时并不需要将其他分支上的全部提交节点统统进行重演。则可以使用 `git rebase --onto` 来选择需要重演的提交节点。

```
git rebase --onto master 5751363
```



NOTE：上面的红色的节点，未被重建（被丢弃）。

rebase 与 merge 区别



`rebase` 会产生线性的提交历史，`merge` 则会产生多个不同分支的合并节点。所以具体没有好坏之分，可根据使用的需求来决定。

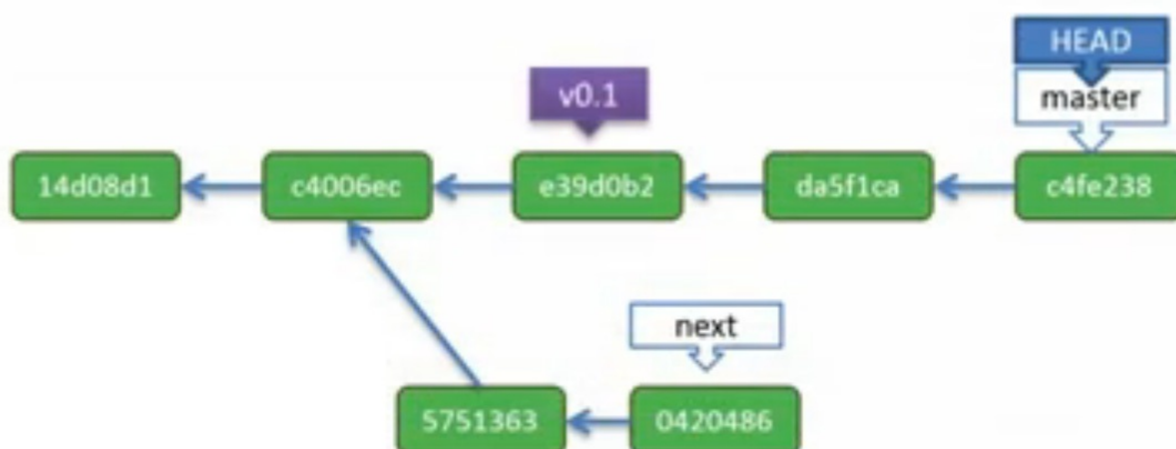
注意！不要在共有分支上使用 `rebase` （例如 `master` 分支）这会导致其他开发者在进行拉取（Pull）时，必须进行合并且合并中包含重复的提交。

tag

不论是 Branch 还是 HEAD 它们均为动态指针，如果想定义一个静止的标示则可以使用 `git tag` ，它将给发布的提交版本设置一个别名。在设置了标签后就可以直接使用标签名来代替它所指代的版本提交了。

```
git tag v0.1 e39d9b2

git checkout v0.1
```



远程操作

远程操作可以将本地仓库推送至远程仓库服务器。**Git** 支持许多主流的通信协议，其中包括 `Local` 、 `HTTP` 、 `SSH` 、还有 `Git` 。服务器只应该是作为同步之用（被动接受既可）。

初始化一个本地的远程服务器

```
git init ~/git-server --bare
```

```
1. X@TimeMachine: ~/Desktop/Git-In-Action (zsh)
X@TimeMachine  ~/Desktop/Git-In-Action  master  git init ~/git-server --bare
Reinitialized existing Git repository in /Users/X/git-server/
X@TimeMachine  ~/Desktop/Git-In-Action  master  tree ~/git-server
/Users/X/git-server
├── HEAD
├── config
├── description
├── hooks
│   ├── applypatch-msg.sample
│   ├── commit-msg.sample
│   ├── post-update.sample
│   ├── pre-applypatch.sample
│   ├── pre-commit.sample
│   ├── pre-push.sample
│   ├── pre-rebase.sample
│   ├── prepare-commit-msg.sample
│   └── update.sample
├── info
│   └── exclude
├── objects
│   ├── info
│   └── pack
└── refs
    ├── heads
    └── tags

8 directories, 13 files
X@TimeMachine  ~/Desktop/Git-In-Action  master  |
```

推送

`git push` 可以将当期的全部版本提交提交推送至远程仓库，其完成了提交历史的完全不复制并同时移动复制版本的 HEAD 与 Branch。

```
1. X@TimeMachine: ~/Desktop/Git-In-Action (zsh)
X@TimeMachine  ~/Desktop/Git-In-Action  master  pwd
/Users/X/Desktop/Git-In-Action
X@TimeMachine  ~/Desktop/Git-In-Action  master  git push ~/git-server master
Counting objects: 15, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (15/15), 1.16 KiB | 0 bytes/s, done.
Total 15 (delta 1), reused 0 (delta 0)
To /Users/X/git-server
 * [new branch]      master -> master
X@TimeMachine  ~/Desktop/Git-In-Action  master  |
```

添加远程仓库别名

`git remote` 可用于添加远程仓库的别名。

```
1. X@TimeMachine: ~/Desktop/Git-In-Action (zsh)
X@TimeMachine  ~/Desktop/Git-In-Action  master  pwd
/Users/X/Desktop/Git-In-Action
X@TimeMachine  ~/Desktop/Git-In-Action  master  git push ~/git-server master
Counting objects: 15, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (15/15), 1.16 KiB | 0 bytes/s, done.
Total 15 (delta 1), reused 0 (delta 0)
To /Users/X/git-server
 * [new branch]      master -> master
X@TimeMachine  ~/Desktop/Git-In-Action  master  git remote add origin ~/git-server
X@TimeMachine  ~/Desktop/Git-In-Action  master  git remote -v
origin  /Users/X/git-server (fetch)
origin  /Users/X/git-server (push)
X@TimeMachine  ~/Desktop/Git-In-Action  master  more .git/config
[core]
        repositoryformatversion = 0
        filemode = true
        bare = false
        logallrefupdates = true
        ignorecase = true
        precomposeunicode = true
[remote "origin"]
        url = /Users/X/git-server
        fetch = +refs/heads/*:refs/remotes/origin/*
X@TimeMachine  ~/Desktop/Git-In-Action  master  |
```

```
# 更改仓库 url 地址
git remote set-url origin 'https://github.com/li-xinyang/FEND_Note.git'
```

远程 **push** 冲突

可以先使用 `git fetch` + `git merge` 来解决冲突的问题。 `git pull` 就等同于 `fetch` 与 `merge` 的合并。

克隆远程仓库

使用 `git clone` 可以克隆远程仓库，并将克隆地址默认设为 `origin` 。

```
1. X@TimeMachine: ~/Desktop/FE_AngularJS (zsh)
 X@TimeMachine  ~/Desktop  git clone https://github.com/li-xinyang/FE_AngularJS.git
Cloning into 'FE_AngularJS'...
remote: Counting objects: 45, done.
remote: Total 45 (delta 0), reused 0 (delta 0), pack-reused 45
Unpacking objects: 100% (45/45), done.
Checking connectivity... done.
 X@TimeMachine  ~/Desktop  cd FE_AngularJS
 X@TimeMachine  ~/Desktop/FE_AngularJS  master  git remote -v
origin  https://github.com/li-xinyang/FE_AngularJS.git (fetch)
origin  https://github.com/li-xinyang/FE_AngularJS.git (push)
 X@TimeMachine  ~/Desktop/FE_AngularJS  master  |
```

技术选择

模块化

NOTE：以下讨论都是基于 JavaScript 的模块组织（每个模块均以文件形式组织），而非工程的模块化。

> The secret to building large app is never build arge apps. Break your applications into small pieces. Then, assemble those testable, bite-sized pieces into your big application.
>
> Justin Meyer

其他语言中的模块支持

- Java - `import`
- C# - `using`
- css - `@import`

但在 JavaScript 中并不存在模块组织在并不支持，于是产生了很多，模块系统。

模块的职责

- 封装实现（将复杂的内容于外界个例）
- 暴露接口（外部可通过接口使用模块）
- 声明依赖（提供给模块系统使用）

模块的使用

反模式（**Anti-Pattern**）

反模式既没有使用任何设计模式。

math.js

```
function add(a, b) {
  return a + b;
}
function sub(a, b) {
  return a - b;
}
```

上面的代码有下面的几个缺点：

- 无封装性
- 接口结构不明显

calculator.js

```
var action = 'add';
function compute(a, b) {
  switch (action) {
    case 'add': return add(a, b);
    case 'sub': return sub(a, b);
  }
}
```

上面的代码也有几个缺点：

- 没有依赖声明
- 使用全局状态

字面量（**Object Literal**）

math.js

```
var math = {
  add: function(a, b) {
    return a + b;
  },
  sub: function(a, b) {
    return a - b;
  }
};
```

结构性好，但没有访问控制。

calculator.js

```
var calculator = {
  action: 'add',
  compute: function(a, b) {
    switch (action) {
      case 'add': return add(a, b);
      case 'sub': return sub(a, b);
    }
  }
}
```

同样没有依赖声明

IIFE（Immediately-invoked Function Expresion)

其为自执行函数。

版本一

calculator.js

```
var calculator = (function(){
  var action = 'add';
  return {
    compute: function(a, b) {
      switch (action) {
        case 'add': return add(a, b);
        case 'sub': return sub(a, b);
      }
    }
  }
})();
```

上面的代码可以进行访问控制，但是不能进行依赖声明。

版本二

calculator.js

```
var calculator = (function(m){
  var action = 'add';
  function compute(a, b) {
    switch (action) {
      case 'add': return m.add(a, b);
      case 'sub': return m.sub(a, b);
    }
  }
  return {
    compute: compute;
  }
})(math)
```

上面的代码虽然可以显示的声明依赖，但是仍然污染了全局变量，而且必须手动进行依赖管理。

命名空间（Namespace）

命运空间可以解决全局变量的污染的问题。

math.js

```
namespace('math', [], function(){
  function add(a, b) { return a + b; }
  function sub(a, b) { return a - b; }
  return {
    add: add,
    sub: sub
  }
})
```

calculator.js

```
//                     依赖声明       依赖注入
//                        |               |
namespace('calculator', ['math'], function(m){
  var action = 'add';
  function compute(a,b) {
    return m[action](a, b);
  }
  return {
    compute: compute;
  }
})
```

模块管理

复杂的模块管理，不能单纯的通过代码文件的排列顺序来进行管理。于是引入了模块系统，它有下面的职责：

- 依赖管理（加载、分析、注入、初始化—）
- 决定模块的写法

常用的模块系统有 `Common.JS` 、 `AMD` 、语言基本的模块化。

CommonJS

CommonJS 是一个模块规范，通常适用于非浏览器环境（NodeJS）。

> A module spec for JavaScript outside the browser.

math.js

```
function add(a, b) {
  return a + b;
}
function sub(a, b) {
  return a - b;
}
exports.add = add;
exports.sub = sub;
```

calculator.js

```
// 依赖声明
var math = require('./math');

function Calculator(container) {
  // ...
}
Calculator.prototype.compute = function(){
  this.result.textContent = math.add(...);
}

// 接口暴露
exports.Calculator = Calculator;
```

优点

- 依赖管理成熟可靠
- 社区活跃且规范接受度高
- 运行时支持且模块化定义简单
- 文件级别的模块作用域隔离
- 可以处理循环依赖

缺点

- 不是标准组织规范
- 同步请求未考虑浏览器环境（可以使用 Browserify 来解决）

```
# browserify 为 npm 下命令行工具
# > 为 Linux/Unix 添加至命令
browserify file0.js > file1.js;
```

打包后的文件如下所示。

```
(function e(t,n,r){function s(o,u){if(!n[o]){if(!t[o]){var a=typeof require=="function"&&require;if(!u&&a)return a(o,!0);if(i)return i(o,!0);var f=new Error("Cannot find module '"+o+"'");throw f.code="MODULE_NOT_FOUND",f}var l=n[o]={exports:{}};t[o][0].call(l.exports,function(e){var n=t[o][1][e];return s(n?n:e)},l,l.exports,e,t,n,r)}return n[o].exports}var i=typeof require=="function"&&require;for(var o=0;o<r.length;o++)s(r[o]);return s})({1:[function(require,module,exports){

},{}]},{},[1]);
```

AMD（Asynchronous Module Definition）

适合异步环境的依赖管理方案。

math.js

```
//   依赖列表
//       |
define([], function(){
  function add(a, b) { return a + b; }
  function sub(a, b) { return a - b; }
// 接口暴露
  return {
    add: add,
    sub: sub
  }
})
```

calculator.js

```
define(['./math'], function(math){
  function Calculator(container) {
    // ...
  }
  Calculator.prototype.compute = function(){
    this.result.textContent = math.add(...);
  };

  // 暴露接口
  return {
    Calculator: Calculator;
  }
})
```

优点

- 依赖管理成熟可靠
- 社区活跃且规范接受度高
- 转为异步环境制作，适合浏览器
- 支持 CommonJS 的书写方式
- 通过插件 API 可以加载非 JavaScript 资源
- 成熟的打包构建工具，并可结合插件一同使用

缺点

- 模块定义繁琐，需要额外嵌套
- 酷基本的支持，需要引入额外的库
- 无法处理循环依赖
- 无法实现条件加载

Simplified CommonJS Wrapping

使用同样的 CommonJS 的依赖管理书写方法，之后在使用正则表达式来提取依赖列表。

```
define(function(require, exports){
  // 依赖声明
  var math = require('./math');

  function Calculator(container) {
    // ...
  }
  Calculator.prototype.compute = function(){
    this.result.textContent = math.add(...);
  }

  // 接口暴露
  exports.Calculator = Calculator;
})
```

Loader Plugins

允许调用处理脚本外的其他资源（例如 HTML 与 CSS 文件），这样就可以形成一个完整的组件。

```
完整组件 = 结构 + 逻辑 + 样式
```

ECMAScript 6 Module

ECMAScript 6 中的模块化管理。

math.js

```
function add(a, b) {
  return a + b;
}
function sub(a, b) {
  return a- b;
}
// export 关键字暴漏接口
export {add, sub}
```

calculator.js

```
import {add} from './math';


class Calculator {
  constructor(container) {}
  compute(){
    this.result.textContent = add(+this.left.value, +this.right.
value);
  }
}
export{Calculator}
```

优点

- 真正的规范未来标准
- 语言基本支持
- 适用于所有的 JavaScript 允许环境
- 可用于处理循环依赖

缺点

- 规范未达到稳定级别
- 暂无浏览器支持

SystemJS

SystemJS 是一个动态模块加载器，下面是它的一下特性：

- 支持加载 AMD
- 支持加载 CommonJS
- 支持加载 ES6
- 支持加载 Transpiler 也可支持任意类型资源

模块管理的对比

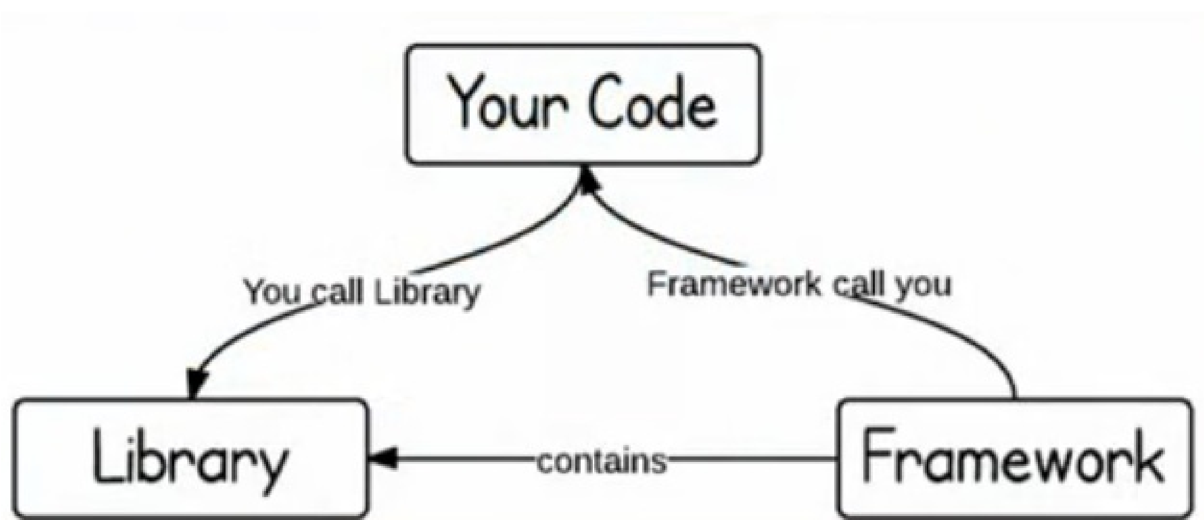
- IIFE，没有解决核心的依赖分析和注入的问题。
- AMD，可以直接使用，库基本的支持。
- CommonJS，可以直接使用，在运行时的支持。
- ES6，语言本身的支持。

使用插件工具，可以将后三种模块管理系统进行相互转换。

框架

NOTE：以下讨论都是基于 JavaScript 的框架。

库（**Library**）与框架（**Framework**）的区别



库 为针对特定问题的解答具有专业性，不控制应用的流程且被动调用。框架 具有控制翻转，决定应用的生命周期，于是便集成了大量的库。

解决方案

常见的解决方案针对的方面：

- DOM
- Communication
- Utility
- Templating
- Component
- Routing（单页系统中尤其重要）
- Architecture

使用外部专业解决方案的原因 可以提高开发效率，可靠性高（浏览器兼容，测试覆盖），也配备优良的配套（文档及工具）。如果外部框架的质量可可靠性无法保证或无法满足业务学期时则不应该选择外部的框架。

实际项目中的使用

- 开发式：基于外部模块系统自由组合
- 半开放：基于一个定制的模块系统，内部外部解决方案共存
- 封闭式：深度定制的模块系统不引入外部模块

DOM

与其相关的有 *Selector*、*Manipulation*、*Event*（*DOM*）、*Animation*。 它的主要职责则为为下面的这些：

- 提供便利的 DOM 查询、操作、移动等操作
- 提供事件绑定及事件代理支持
- 提供浏览器特性检测及 UserAgent 侦测
- 提供节点属性、样式、类名的操作
- 保证目标平台的跨浏览器支持

常用的 DOM 库有 **jQuery**（使用链式接口），**zepto.JS**，**MOOTOO.JS**（使用原生 DOM 对象，通过直接跨站了 DOM 原生对象）。

基础领域

库名	大小	兼容性	优点	缺点
MOOTOO.JS	96KB	IE6+	概念清晰、无包装对象、接口设计优秀、源码清晰易懂、不局限于 DOM 与 AJAX	扩展原生对象（致命）、社区衰弱
jQuery	94KB	IE6+	社区强大普及率高、包装对象、专注于 DOM	包装对象（容易混淆）
zepto.JS	25KB	IE10+	小且启动快、接口与 jQuery 兼容、提供简单手势操作	无法与 jQuery 100% 对于、支持浏览器少、功能弱

专业领域

领域	库名	大小	描述
手势	Hammer.JS	12KB	常见手势封装（Tab、Hold、Transform、Swifp）并支持自定义
高级动画	Velocity.JS	12KB	复杂动画序列实现，不仅局限于 DOM
视频播放	Video.JS	101KB	类似原生 video 标签的使用方式，对低级浏览器使用 flash 播放器
局部滚动	isscroll.JS	13KB	移动端 `position:fix` + `overflow:scroll` 的救星

Communication

与其相关的有 *XMLHttpRequest*、*Form*、*JSONP*、*Socket*。 它的主要职责则为为下面的这些：

- 处理与服务器的请求与相应
- 预处理请求数据与响应数据 Error/Success 的判断封装
- 多类型请求，统一接口（XMLHttpRequest1/2、JSONP、iFrame）
- 处理浏览器兼容性

库名	大小	支持
Reqwest	3.4KB	JSONP支持、稳定 IE6+支持、CORS 跨域、Promise/A 支持
qwest	2.5KB	代码少、支持XMLHttpRequest2、CORS 跨域、支持高级数据类型（ArrayBuffer、Blob、FormData）

实时性要求高的需求

库名	支持
socket.io	实时性、支持二进制数据流、智能自动回退支持、支持多种后端语言（NodeJS 最为稳妥）

Utility（Lang）

与其相关的有 函数增强 *& Shim*（保证实现与规范一致）、*Flow Control*。 它的主要职责则为为下面的这些：

- 提供 JavaScript 原生不提供的功能
- 方法门面包装使其便于使用

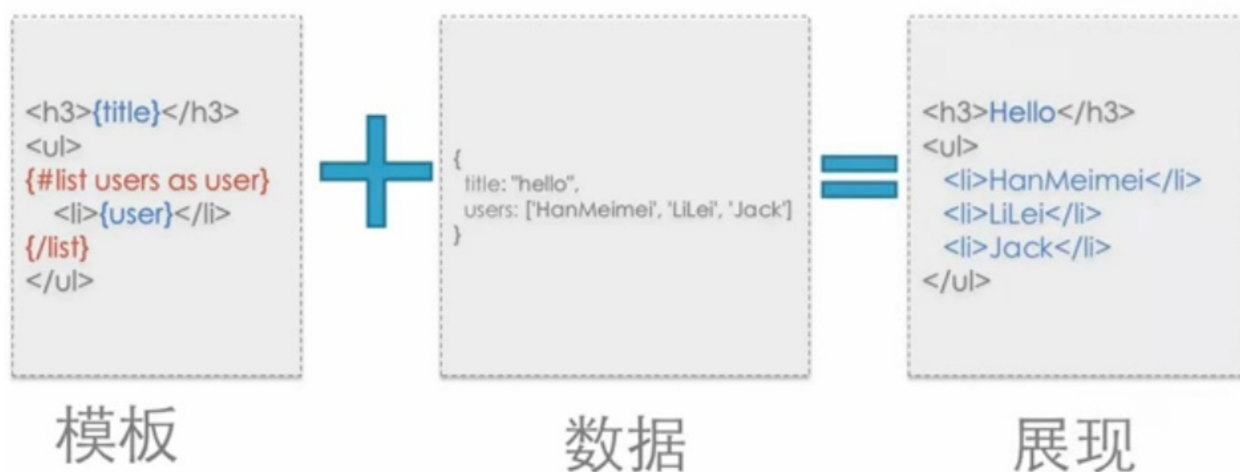
- 异步列队及流程控制

库名	大小	描述
es5-shim	53KB	提供 ES3 环境下的 ES5 支持
es6-shim	38KB	
underscore	16.5KB	兼容 IE6+ 的扩展功能函数
Lodash	50KB	其为 underscore 的高性能版本，方法多为 runtime 编译出来的

Templating

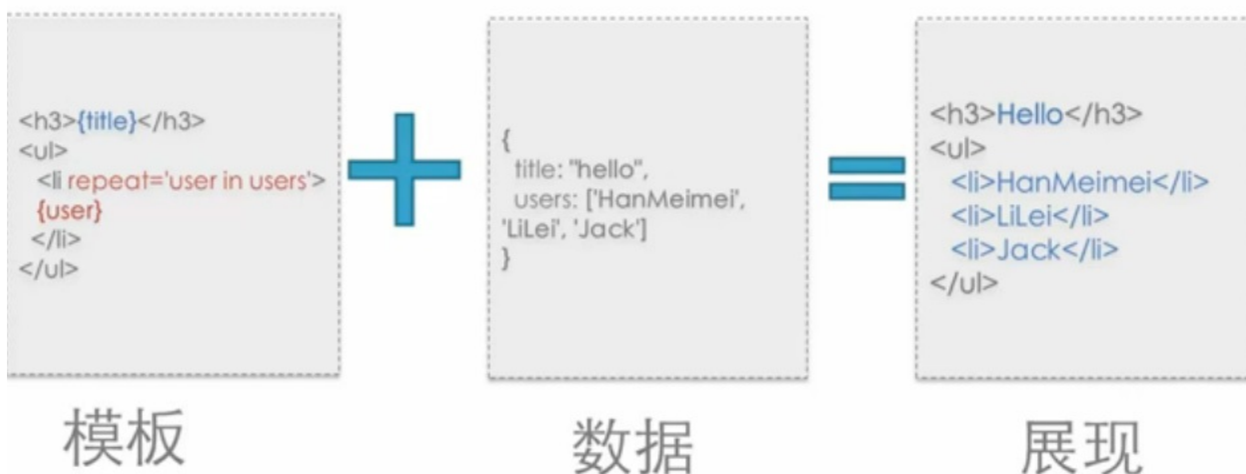
与其相关的有 *String-based*、*DOM-based*、*Living Template*。

基于字符串的模板



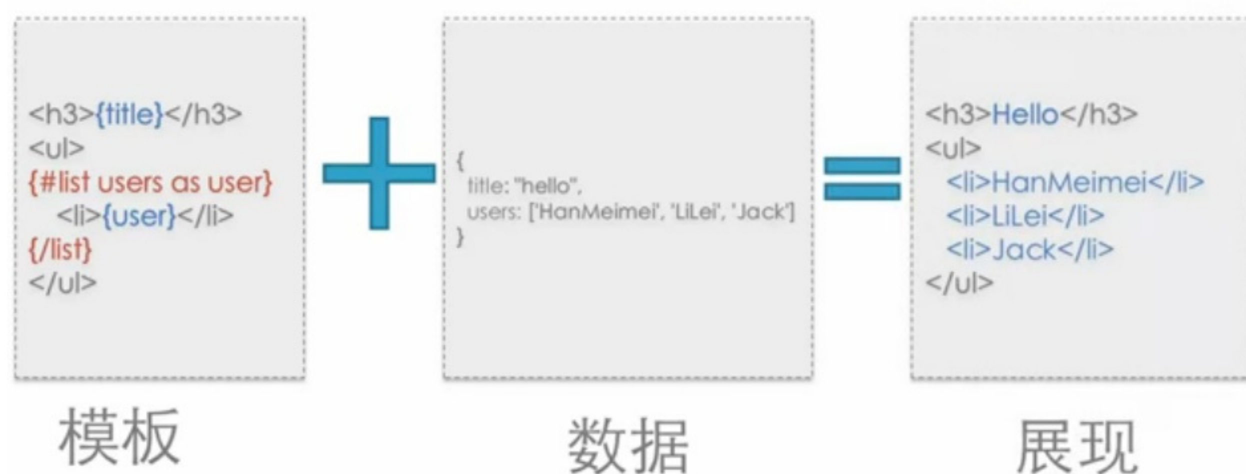
之后的数据修改展现不会进行变化，如果重新绘制（性能低）页面则会去除已有的 DOM 事件。

基于 **DOM** 的模板



修改数据可以改变显示（性能更好）也会保留 DOM 中的已有事件，最终导致 DOM 树与数据模型相联系。

Living-Template



其拼接了字符串模板和 DOM 模板的技术（类似 Knockout.JS 注释的实现），最终导致 DOM 树与数据模型相联系。

	String-based	**DOM-based**	**Living-Template**
好处	可以服务器端运行		
解决方案	dust.JS、hogan、dot.JS	Angular.JS、Vue.JS、Knockout	Regular.JS、Ractive.JS、htmlbar
初始化时间	☆☆☆	☆	☆☆
动态更新	无	☆☆☆	☆☆☆
DOM 无关	☆☆☆	无	☆☆
语法	☆☆☆	☆	☆☆
学习成本	☆	☆☆☆	☆☆
SVG 支持	无	☆☆	☆☆
安全性	☆	☆	☆☆☆

Component

与其相关的有 *Modal*、*Slider*、*DatePicker*、*Tabs*、*Editor*（其为产品开发中最耗时也是最必要的一部分）。它的主要职责则为为下面的这些：

- 提供基础的 CSS 支持
- 提供常见的组件
- 提供声明式的调用方式（类似 Bootstrap）

组件库名	版本	特定	支持
Bootstrap	3.x	Mobile First 流式栅格，基于 LESS与 SASS 组织可定制 UI，提供大量组件	IE8+
Foundation	5.x	Mobile First 流式栅格，基于 SASS 组织，可定制 UI，提供大量组件	IE9+

NOTE：有存在不使用 jQuery 版本的 Bootstrap 可供使用。

Router

与其相关的有 *Client Side*、*Server Side*。它的主要职责则为为下面的这些：

- 监听 URL 变化，并通知注册的模块
- 通过 JavaScript 进行主动跳转
- 历史管理
- 对目标浏览器的兼容性支持

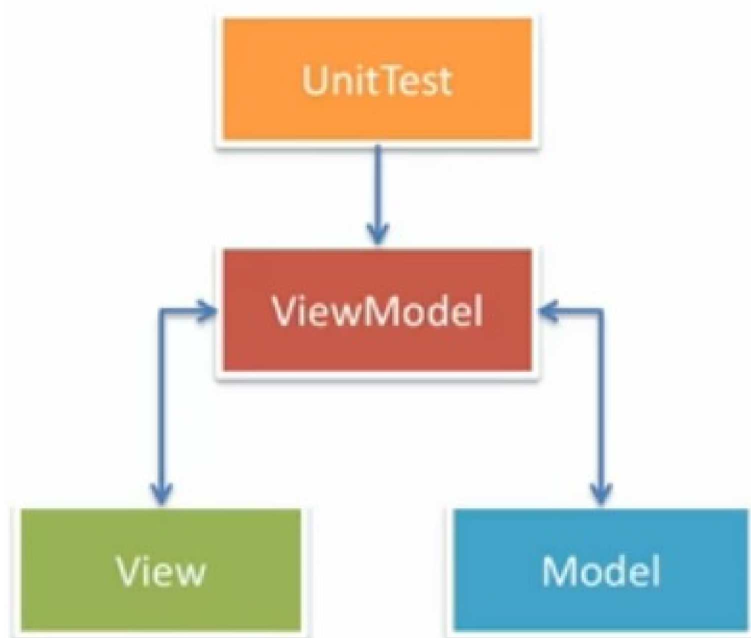
路由库名	大小	特定	支持
page.JS	6.2KB	类似 Express.Router 的路由规则的前端路由库	IE8+
Director.JS	10KB	可以前后端使用同一套规则定义路由	IE6+
Stateman	10KB	处理深层复杂路由的独立路优库	IE6+
crossroad.JS	7.5KB	老牌路由库，API 功能较为繁琐	

Architecture（解耦）

与其相关的有 *MVC*、*MVVC*、*MV**，解耦又可以通过很多方式来实现（例如事件、分层）。它的主要职责则为为下面的这些：

- 提供一种凡是帮助（强制）开发者进行模块解耦
- 视图与模型分离
- 容易进行单元测试
- 容易实现应用扩展

下面以 **MVVM**为例：



- Model 数据实体用于记录应用程序的数据
- View 友好的界面其为数据定制的反映，它包含样式结构定义以及 VM 享有的声明式数据以及数据绑定
- ViewModel 其为 View 与 Model 的粘合，它通过绑定事件与 View 交互并可以调用 Service 处理数据持久化，也可以通过数据绑定将 Model 的变动反映到 View 中

NOTE：MV 不等同于 *SPA*，路由是 *MV* 系统的课定位状态信息来源。 NOTE+：单页系统的普世法则为可定位的应用程序状态都应该统一由路由系统进入，以避免网状的信息流。 NOTE++：库与框架选择站点microjs javascriptOO JavaScripting

开发实践

系统设计

NOTE：综合运用实习案例，本章使用案例为网易云音乐， 并且主要关注前端工程师的工作职责，其他工程师的职责规范并不包含。

交互流程说明

通过交互文案来了解用户行为与异常提示。

系统分解

例如下面的独立的子系统：

- 注册登录密码
- 系统主框架
  - 顶栏
    - 搜索
    - 账号
    - 消息
    - 设置
  - 边栏
    - 歌单操作
    - 其他
  - 底栏
    - 播放器
    - 播放列表
    - 歌曲详情
  - 内容区

系统分解必须对照交互稿做到百分之百的对应，不能漏掉任何一个模块。 后续的开发与评估都需根据此分解进行。

接口设计

分析模块交互理解需求以及交互行为。对于数据获取的形式进行适当的假设， 并定义数据类型、模板资源、异步接口、以及页面摘要。

项目结构

根据规范说明就可以做出项目的结构定义，项目结构分两部分后端模板与前端实现。

初始代码

前端工程师需要在开发工具中添加开发规范，以及测试使用的模拟数据。

系统实现

视觉说明（视觉稿）中包含各个情况下用户界面的显示样式，其定义了交互稿中的所有效果。之后则需要从中提取出通用组件，其中包括：

- 通用原件（Logo，提示，输入框，图标，按钮等）
- 通用列表
- 复合组件（留言板类）
- 浮层弹出

测试发布

使用同步模拟数据进行 本地测试 测试，只需在入口页面引入既可。异步处理则可以使用第三方的代理软件既可。在本地测试完后，需要前后端的对接联调。去除模拟同步数据，直接使用后端数据既可。然而对于异步数据联调直接去除代理转向既可。

在完成测试后需要打包发布上线，可使用自动化工具将工程文件打包优化（合并压缩混淆）。

Table of Contents *generated with DocToc*

书单

HTML

N/A

CSS

- CSS Mastery: Advanced Web Standard Solutions

JavaScript

- Professional JavaScript for Web Developers
- DOM Scripting: Web Design with JavaScript and the Document Object Model
- AdvancED DOM Scripting: Dynamic Web Design Techniques
- JavaScript: The Definitive Guide

Version Control

- Git 简明指南
- Git Pro